

The African School
of Fundamental
Physics and
Applications



Integrating Scientific Computing into Math and Science Classes

Dave Biersach
Brookhaven National
Laboratory
dbiersach@bnl.gov



Session 03
Computing in Science

Session 03 – Topics

- Learn about Python's **dictionary** data structure and how to read data files in **CSV** and **JSON** formats
- Analyze chemical trends and anomalies across the **Periodic Table of Elements**
- Implement the **linear regression** formulas to find the **line of best fit** using the method of least squares created by Gauss
- Identify an *unknown element* using the **Ideal gas law**
- Determine the height and velocity of **cosmic ray** showers using Newtonian Kinematics
- Simulate the **trajectory** of a circus cannon performer to determine a safe initial launch velocity

Run python_dictionaries.ipynb – Cells 1...2

Import the pretty printer module from the `pprint` package

```
[1] # Cell 1
    from pprint import pprint ← ①
```

Create a dictionary (key/value pairs) of nations and their capitals

1. The keys will be of type `string`, and only one key will exist for each value
2. The values will be of type `string`, and only one value will exist for each key

However, Python dictionaries can have multiple items with a key or value
But you can never have repeated keys - all keys must be unique!

```
[2] # Cell 2
    capitals = {
        "Morocco": "Rabat", ← ②
        "Germany": "Berlin",
        "France": "Paris",
        "South Africa": "Pretoria",
        "Russia": "Moscow",
        "Egypt": "Cairo",
        "USA": "Washington D.C.",
        "India": "New Delhi",
        "China": "Beijing",
    }
```

Run python_dictionaries.ipynb – Cells 3...4

```
"Pretty" print the capitals dictionary
Notice the pretty printer displays the dictionary sorted by keys

[3] # Cell 3 ← ①
    pprint(capitals)

↵ {'China': 'Beijing',
   'Egypt': 'Cairo', ← ②
   'France': 'Paris',
   'Germany': 'Berlin',
   'India': 'New Delhi',
   'Morocco': 'Rabat',
   'Russia': 'Moscow',
   'South Africa': 'Pretoria',
   'USA': 'Washington D.C.'}

Read the value of a specific key

[4] # Cell 4
    print(capitals["USA"]) ← ③

↵ Washington D.C. ← ④
```

Run python_dictionaries.ipynb – Cells 5..6

Create/insert new key/value pairs into the dictionary using two different methods/syntax

```
[5] # Cell 5
capitals.update({"United Kingdom": "London"}) ← ①
capitals["Spain"] = "Madrid"
pprint(capitals)
```

```
{'China': 'Beijing',
 'Egypt': 'Cairo',
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'Russia': 'Moscow',
 'South Africa': 'Pretoria',
 'Spain': 'Madrid', ← ②
 'USA': 'Washington D.C.',
 'United Kingdom': 'London'}
```

Update/change/edit the value for a key that already exists in the dictionary

```
# Cell 6
capitals["USA"] = "New York" ← ③
pprint(capitals)
```

```
{'China': 'Beijing',
 'Egypt': 'Cairo',
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'Russia': 'Moscow',
 'South Africa': 'Pretoria',
 'Spain': 'Madrid',
 'USA': 'New York', ← ④
 'United Kingdom': 'London'}
```

Run python_dictionaries.ipynb – Cell 7

```
Delete/remove a key from the dictionary

[7] # Cell 7
del capitals["Russia"]
pprint(capitals)
```

```
{'China': 'Beijing',
 'Egypt': 'Cairo',
 'France': 'Paris',
 'Germany': 'Berlin',
 'India': 'New Delhi',
 'Morocco': 'Rabat',
 'South Africa': 'Pretoria',
 'Spain': 'Madrid',
 'USA': 'New York',
 'United Kingdom': 'London'}
```

①

②

Liquid Range

- Your scientist has asked you to plot the **melting** and **boiling** point of each element across the periodic chart
- She is interested in visualizing any **trends** and *discontinuities* in elements having the same valence shell structure
- Therefore, she wants you to plot the elements first **by group**, then **by period**, then **by atomic number** order
- She has asked you to include elements from both the **Lanthanoid** and **Actinoid** series
- Is there a readily available **data set** on the web that encodes the entire periodic chart?

The Periodic Table (circa 2021)

period	group																	18	
	1*																	2	
	1	1																	18
	2	3	4											5	6	7	8	9	10
	3	11	12											13	14	15	16	17	18
	4	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
	5	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
6	55	56	57	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	
7	87	88	89	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	

- Alkali metals
- Alkaline-earth metals
- Transition metals
- Other metals
- Other nonmetals
- Halogens
- Noble gases
- Rare-earth elements (21, 39, 57–71) and lanthanoid elements (57–71 only)
- Actinoid elements

lanthanoid series	6	58	59	60	61	62	63	64	65	66	67	68	69	70	71
		Ce	Pr	Nd	Pm	Sm	Eu	Gd	Tb	Dy	Ho	Er	Tm	Yb	Lu
		140.116	140.90766	144.242	(145)	150.36	151.964	157.25	158.925354	162.5	164.930328	167.259	168.934218	173.045	174.9668
actinoid series	7	90	91	92	93	94	95	96	97	98	99	100	101	102	103
		Th	Pa	U	Np	Pu	Am	Cm	Bk	Cf	Es	Fm	Md	No	Lr
		232.0377	231.03588	238.02891	(237)	(244)	(243)	(247)	(247)	(251)	(252)	(257)	(258)	(259)	(262)

The Periodic Table (circa 2021)

- Alkali metals
- Alkaline-earth metals
- Transition metals
- Other metals
- Other nonmetals
- Halogens
- Noble gases
- Rare-earth elements (21, 39, 57–71) and lanthanoid elements (57–71 only)
- Actinoid elements

period	group																		
1	1*																		2
1	1											13	14	15	16	17	18		
2	2											5	6	7	8	9	10		
3	3											13	14	15	16	17	18		
4	4											31	32	33	34	35	36		
5	5											49	50	51	52	53	54		
6	6											81	82	83	84	85	86		
7	7											113	114	115	116	117	118		
		19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
		37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
		55	56	57	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86
		87	88	89	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118
		(223)	(226)	(227)	(261)	(262)	(266)	(264)	(277)	(268)	(281)	(280)	(285)	(286)	(289)	(288)	(293)	(294)	(294)

All elements in a **group** have the same number of valence *electrons*. As a result, elements in the same group often display similar properties and *reactivity*.

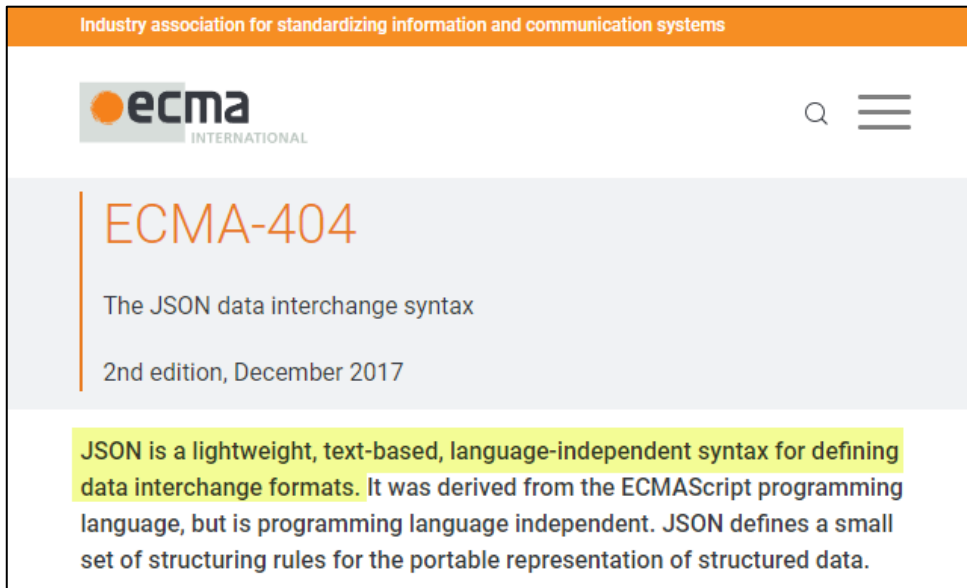
All elements in a **period** have the same number of electron shells. Each *next* element in a period **has one more proton** and is *less* metallic than its predecessor.

Python Dictionaries and JSON Data Files

JavaScript **Object Notation** (JSON)

"Key-Value Pairs"

<https://www.json.org>



Industry association for standardizing information and communication systems

ecma
INTERNATIONAL

ECMA-404

The JSON data interchange syntax

2nd edition, December 2017

JSON is a lightweight, text-based, language-independent syntax for defining data interchange formats. It was derived from the ECMAScript programming language, but is programming language independent. JSON defines a small set of structuring rules for the portable representation of structured data.


Python Dictionaries and JSON Data Files

JavaScript **Object Notation** (JSON)

"Key-Value Pairs"

<https://www.json.org>

Industry association for standardizing information and communication systems



ECMA-404

The JSON data interchange syntax

2nd edition, December 2017

JSON is a lightweight, text-based, language-independent data interchange format. It was derived from JavaScript, a programming language, but is programming language independent. It is a set of structuring rules for the portable representation of structured data.

<https://realpython.com/python-json>

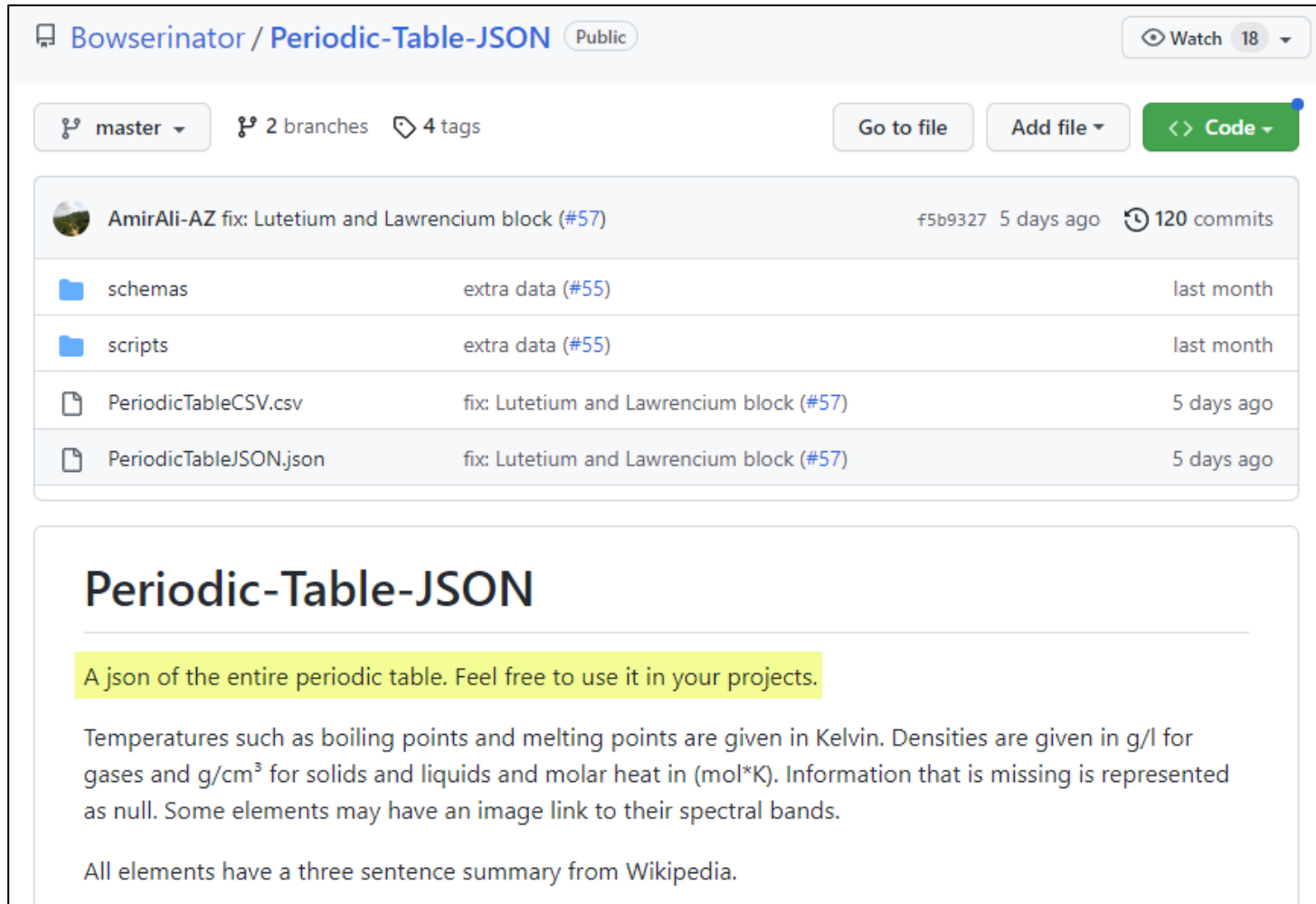
Working With JSON Data in Python

Table of Contents

- A (Very) Brief History of JSON
- Look, it's JSON!
- Python Supports JSON Natively!
 - A Little Vocabulary
 - Serializing JSON
 - A Simple Serialization Example
 - Some Useful Keyword Arguments
 - Deserializing JSON

The Periodic Table as JSON

<https://github.com/Bowserinator/Periodic-Table-JSON>



The screenshot shows the GitHub interface for the repository 'Bowserinator / Periodic-Table-JSON'. At the top, it indicates the repository is 'Public' and has 18 watchers. Below this, there are navigation options for 'master' (selected), '2 branches', and '4 tags'. Action buttons include 'Go to file', 'Add file', and 'Code'. The main content area displays a commit by 'AmirAli-AZ' titled 'fix: Lutetium and Lawrencium block (#57)' with commit hash 'f5b9327' and '120 commits' made '5 days ago'. Below the commit, a list of files is shown:

File Name	Commit	Time
schemas	extra data (#55)	last month
scripts	extra data (#55)	last month
PeriodicTableCSV.csv	fix: Lutetium and Lawrencium block (#57)	5 days ago
PeriodicTableJSON.json	fix: Lutetium and Lawrencium block (#57)	5 days ago

Below the file list, the repository title 'Periodic-Table-JSON' is repeated. A highlighted yellow box contains the text: 'A json of the entire periodic table. Feel free to use it in your projects.' Below this, a paragraph explains: 'Temperatures such as boiling points and melting points are given in Kelvin. Densities are given in g/l for gases and g/cm³ for solids and liquids and molar heat in (mol*K). Information that is missing is represented as null. Some elements may have an image link to their spectral bands.' A final paragraph states: 'All elements have a three sentence summary from Wikipedia.'

periodic_table.json

JSON is like a
Python dictionary with
keys and *values*

```
{
  "name": "Hydrogen",
  "appearance": "colorless gas",
  "atomic_mass": 1.008,
  "boil": 20.271, ← ⑥
  "category": "diatomic nonmetal",
  "density": 0.08988,
  "discovered_by": "Henry Cavendish",
  "melt": 13.99, ← ⑤
  "molar_heat": 28.836,
  "named_by": "Antoine Lavoisier",
  "number": 1, ← ②
  "period": 1, ← ④
  "group": 1, ← ③
  "phase": "Gas",
  "source": "https://en.wikipedia.org/wiki/Hydrogen",
  "bohr_model_image": "https://storage.googleapis.com/search-ar-edu/periodic-table/element_001_hydrogen/",
  "bohr_model_3d": "https://storage.googleapis.com/search-ar-edu/periodic-table/element_001_hydrogen/ele",
  "spectral_img": "https://en.wikipedia.org/wiki/File:Hydrogen_Spectra.jpg",
  "summary": "Hydrogen is a chemical element with chemical symbol H and atomic number 1. With an atomic",
  "symbol": "H", ← ①
  "xpos": 1,
  "ypos": 1,
  "wxpos": 1,
  "wypos": 1,
  "shells": [
    1
  ],
  "electron_configuration": "1s1",
  "electron_configuration_semantic": "1s1",
  "electron_affinity": 72.769,
  "electronegativity_pauling": 2.2,
  "ionization_energies": [
    1312
  ],
  "cpk-hex": "ffffff",
  "image": {
    "title": "Vial of glowing ultrapure hydrogen, H2. Original size in cm: 1 x 5",
    "url": "https://upload.wikimedia.org/wikipedia/commons/d/d9/Hydrogenglow.jpg",
    "attribution": "User:Jurii, CC BY 3.0 <https://creativecommons.org/licenses/by/3.0>, via Wikimedia C
  },
  "block": "s"
}
```

A value can itself be a
"nested" dictionary

Run plot_liquid_range.ipynb – Cells 1...2

Import necessary packages/modules

```
[1] # Cell 1
import json ← ①
from pathlib import Path ← ②

import matplotlib.pyplot as plt
import numpy as np
from google.colab import drive ← ③

import pandas as pd ← ④
```

Connect this notebook to your Google Drive

```
[2] # Cell 2
drive.mount("/content/gdrive", force_remount=True) ← ⑤
notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
notebook_path = notebook_path / Path("Session 03 - Computing in Science") ← ⑥
notebook_path
```

```
↳ Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science') ← ⑦
```

View plot_liquid_range.ipynb – Cell 2

Permit this notebook to access your Google Drive files?


This notebook is requesting access to your Google Drive files. Granting access to Google Drive will permit code executed in the notebook to modify files in your Google Drive. Make sure to review notebook code prior to allowing this access.


No thanks

[Connect to Google Drive](#)




View plot_liquid_range.ipynb – Cell 2

 Sign in with Google



Sign in to Google Drive for desktop


 dbiersach@gmail.com

By continuing, Google will share your name, email address, language preference, and profile picture with Google Drive for desktop. See Google Drive for desktop's [Privacy Policy](#) and [Terms of Service](#).


You can manage Sign in with Google in your [Google Account](#).

View plot_liquid_range.ipynb – Cell 2

Sign in with Google



Google Drive for desktop wants additional access to your Google Account

 dbiersach@gmail.com

i Google Drive for desktop already has some access

See the [8 services](#) that Google Drive for desktop has some access to.

Make sure you trust Google Drive for desktop

You may be sharing sensitive info with this site or app. Learn about how Google Drive for desktop will handle your data by reviewing its [terms of service](#) and [privacy policies](#). You can always see or remove access in your [Google Account](#).

[Learn about the risks](#)

View plot_liquid_range.ipynb – Cell 2

Import necessary packages/modules

```
[1] # Cell 1
import json
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
from google.colab import drive

import pandas as pd
```

Connect this notebook to your Google Drive

```
[2] # Cell 2
drive.mount("/content/gdrive", force_remount=True)
notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
notebook_path = notebook_path / Path("Session 03 - Computing in Science")
notebook_path
```



```
Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')
```

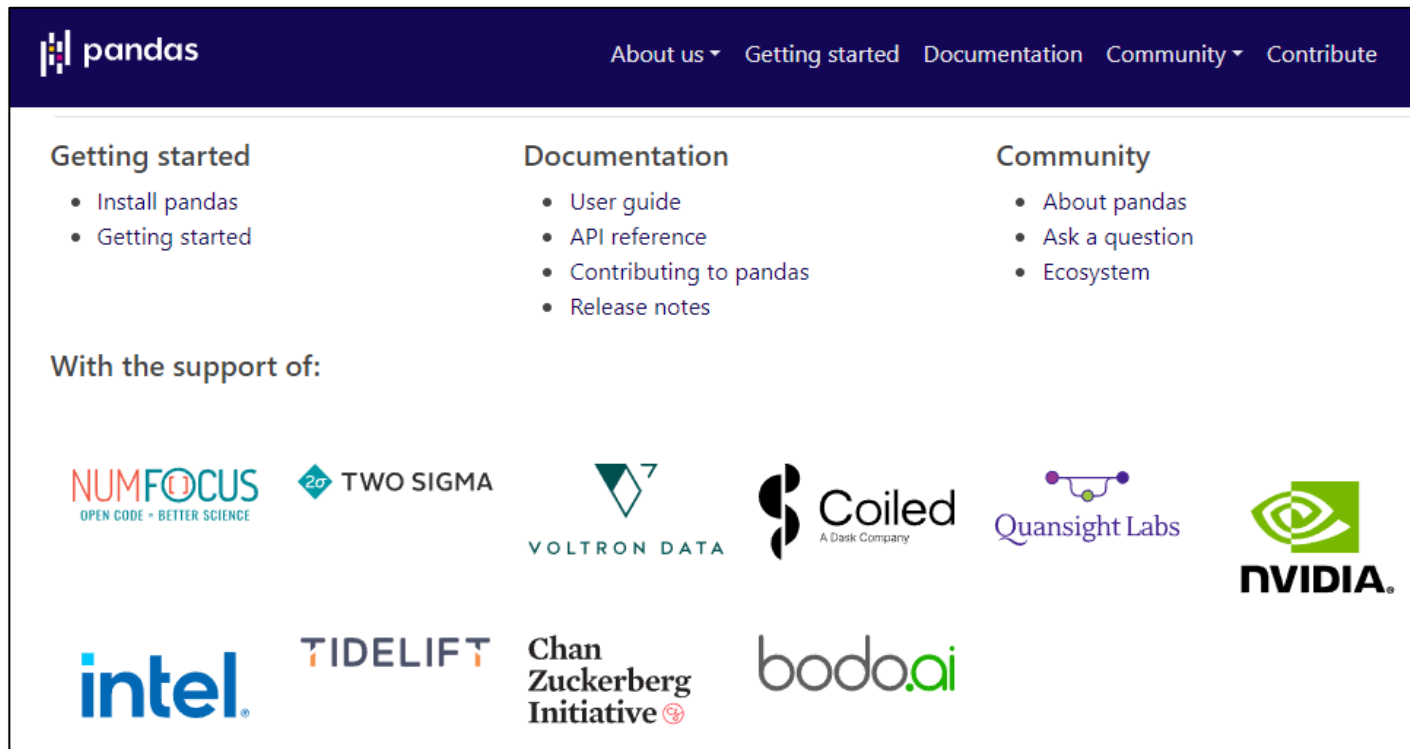
Run plot_liquid_range.ipynb – Cell 3

Rehydrate/deserialize a Python dictionary from a JSON formatted text file

```
[3] # Cell 3
    file_name = "periodic_table.json"
    file_path = notebook_path/ file_name ← ①
    with Path.open(file_path, "rb") as f_in:
        periodic_table = json.load(f_in) ← ②
    periodic_table["elements"][0]["name"] ← ③
    ↵ 'Hydrogen' ← ④
```

Extending Python via the **pandas** Package

<https://pandas.pydata.org>



The screenshot shows the pandas website homepage. At the top, there is a dark blue navigation bar with the pandas logo on the left and links for "About us", "Getting started", "Documentation", "Community", and "Contribute" on the right. Below the navigation bar, the page is divided into three columns: "Getting started", "Documentation", and "Community".

Getting started

- Install pandas
- Getting started

Documentation

- User guide
- API reference
- Contributing to pandas
- Release notes

Community

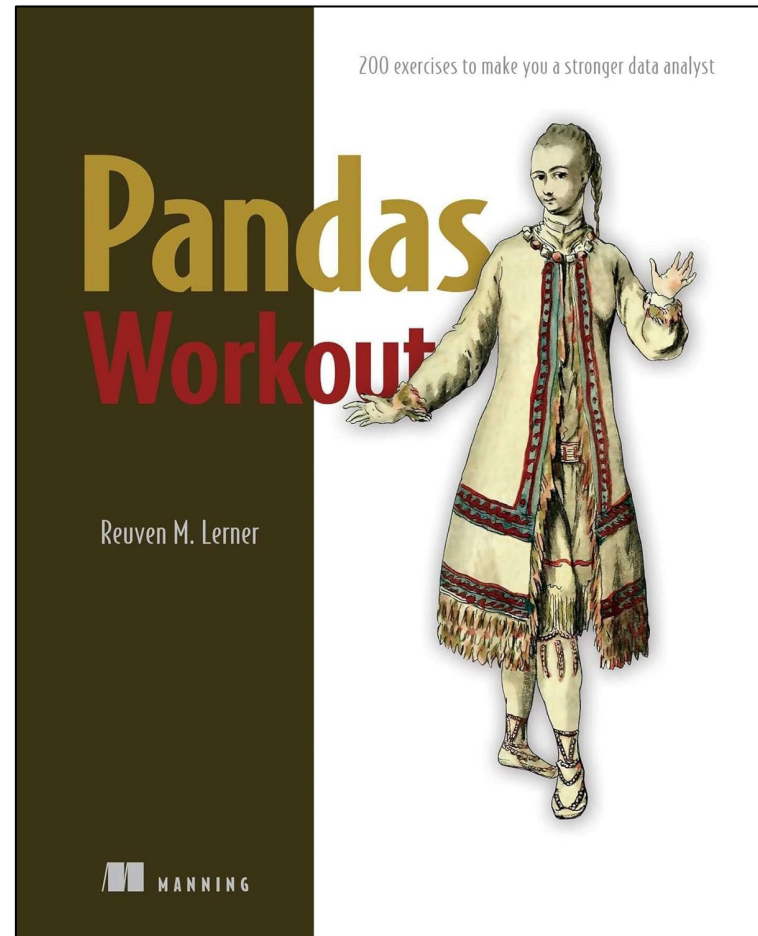
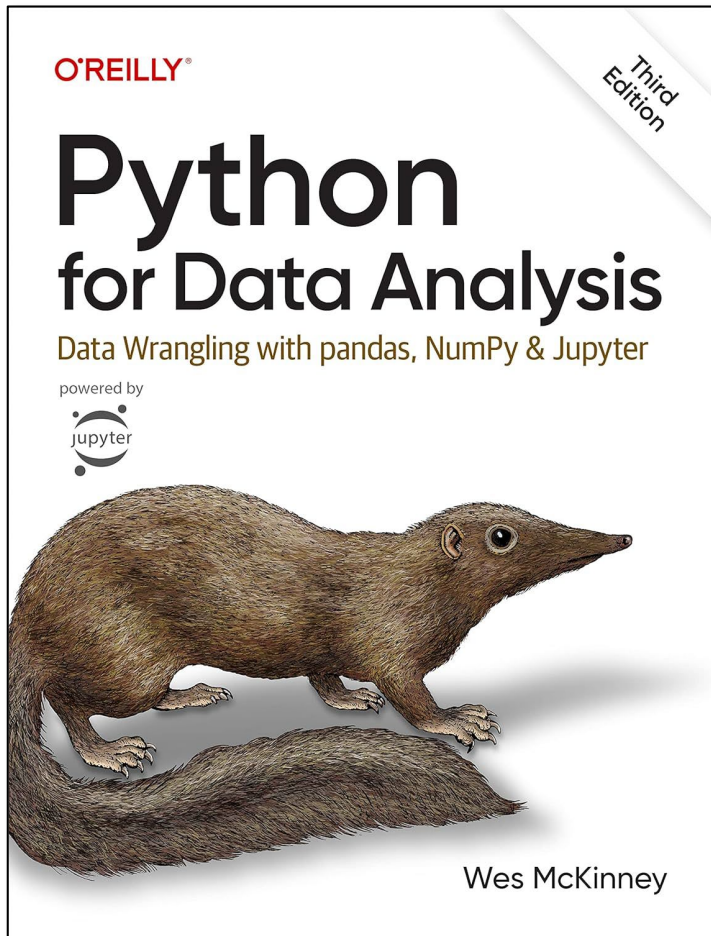
- About pandas
- Ask a question
- Ecosystem

Below these columns, the text "With the support of:" is followed by a grid of logos for various sponsors:

- NUMFOCUS (OPEN CODE - BETTER SCIENCE)
- TWO SIGMA
- VOLTRON DATA
- Coiled (A Dask Company)
- Quansight Labs
- NVIDIA
- intel
- TIDELIFT
- Chan Zuckerberg Initiative
- bodo.ai

Extending Python via the **pandas** Package

Pandas is 100% Open Source and Free of Cost

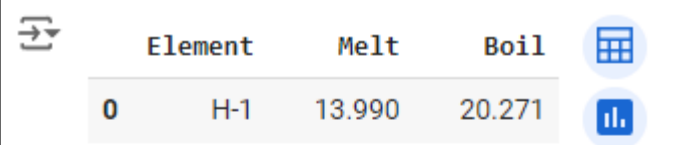


Run plot_liquid_range.ipynb – Cell 4

Create a Python `list` of elements sorted by group, then period, then atomic number ← ①

1. Use `for` loops to range over every group # and period #
2. Extract the dictionary values for those keys (elements) that match the current group # and period #
3. Append the matching element to the growing `list` of elements
4. Each item in the `elements` list is a tuple containing:
 - a. A `string` label formed by of its atomic symbol, a hyphen
 - b. The element's melting point (or $-\infty$ if not melting point
 - c. The element's boiling point (or ∞ if not melting point is

```
[4] # Cell 4
elements: list = [] ← ③
for group in range(1, 19): ← ④
    for period in range(1, 8): ← ⑤
        for k, v in enumerate(periodic_table["elements"]):
            if group == int(v["group"]) and period == v["period"]:
                elements.append(← ⑦
                    (
                        f"{v['symbol']}-{v['number']}",
                        float(v["melt"] or -np.inf),
                        float(v["boil"] or np.inf),
                    ),
                )
pd.DataFrame(elements[:10], columns=["Element", "Melt", "Boil"]) ← ⑩
```



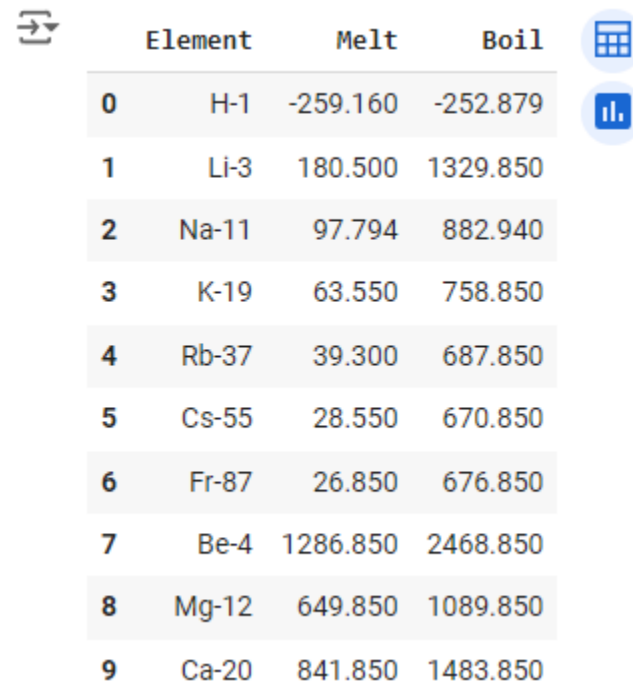
	Element	Melt	Boil
0	H-1	13.990	20.271
1	Li-3	453.650	1603.000
2	Na-11	370.944	1156.090
3	K-19	336.700	1032.000
4	Rb-37	312.450	961.000
5	Cs-55	301.700	944.000
6	Fr-87	300.000	950.000
7	Be-4	1560.000	2742.000
8	Mg-12	923.000	1363.000
9	Ca-20	1115.000	1757.000

Run plot_liquid_range.ipynb – Cell 5

In preparation for plotting the melting and boiling points: ← ①

1. Create a numpy array from the elements list
2. Create a numpy array of the melting points, which are the 2nd item [index #1] in each tuple in the elements list
3. Create a numpy array of the boiling points, which are the 3rd item [index #2] in each tuple in the elements list
4. Convert both melting and boiling point values from Kelvin to Celsius ← ②

```
[5] # Cell 5
data = np.array(elements) ← ③
melt = np.array(data[:, 1], dtype=float) - 273.15
boil = np.array(data[:, 2], dtype=float) - 273.15
pd.DataFrame({ ← ⑤
    'Element' : [element[0] for element in elements[
    'Melt': melt[:10],
    'Boil': boil[:10]
})
```



	Element	Melt	Boil
0	H-1	-259.160	-252.879
1	Li-3	180.500	1329.850
2	Na-11	97.794	882.940
3	K-19	63.550	758.850
4	Rb-37	39.300	687.850
5	Cs-55	28.550	670.850
6	Fr-87	26.850	676.850
7	Be-4	1286.850	2468.850
8	Mg-12	649.850	1089.850
9	Ca-20	841.850	1483.850

Run plot_liquid_range.ipynb – Cells 6...7

Find the element with the smallest liquid range (boiling point - melting point)

1. The variable `liquid_range` becomes a numpy array because *boil* and *melt* are both arrays
2. The function `np.argmin()` returns the index in the given array with the minimum value

```
[6] # Cell 6
    liquid_range = boil - melt
    min_idx = np.argmin(liquid_range) ← ①
    min_range = liquid_range[min_idx]
    print(f"Smallest liquid range: {min_range:,.2f}°C is {data[min_idx, 0]}")
```

```
↔ Smallest liquid range: 2.54°C is Ne-10
```

Find the element with the largest liquid range

1. The `liquid_range` array is sorted so the highest values are first
2. The function `np.argwhere().min()` returns the first index where the condition is true
3. Recall that elements without a known boiling point have their boiling point sent to ∞
4. Therefore, we need to find the first index where the maximum liquid range that is NOT ∞

```
[7] # Cell 7
    measured_liquid_range = np.array(sorted(liquid_range, reverse=True))
    max_measured_idx = np.argwhere(measured_liquid_range < np.inf).min()
    max_range = measured_liquid_range[max_measured_idx] ← ②
    max_idx = np.argwhere(liquid_range == max_range)[0, 0]
    print(f" Largest liquid range: {max_range:,.2f}°C is {data[max_idx, 0]}")
```

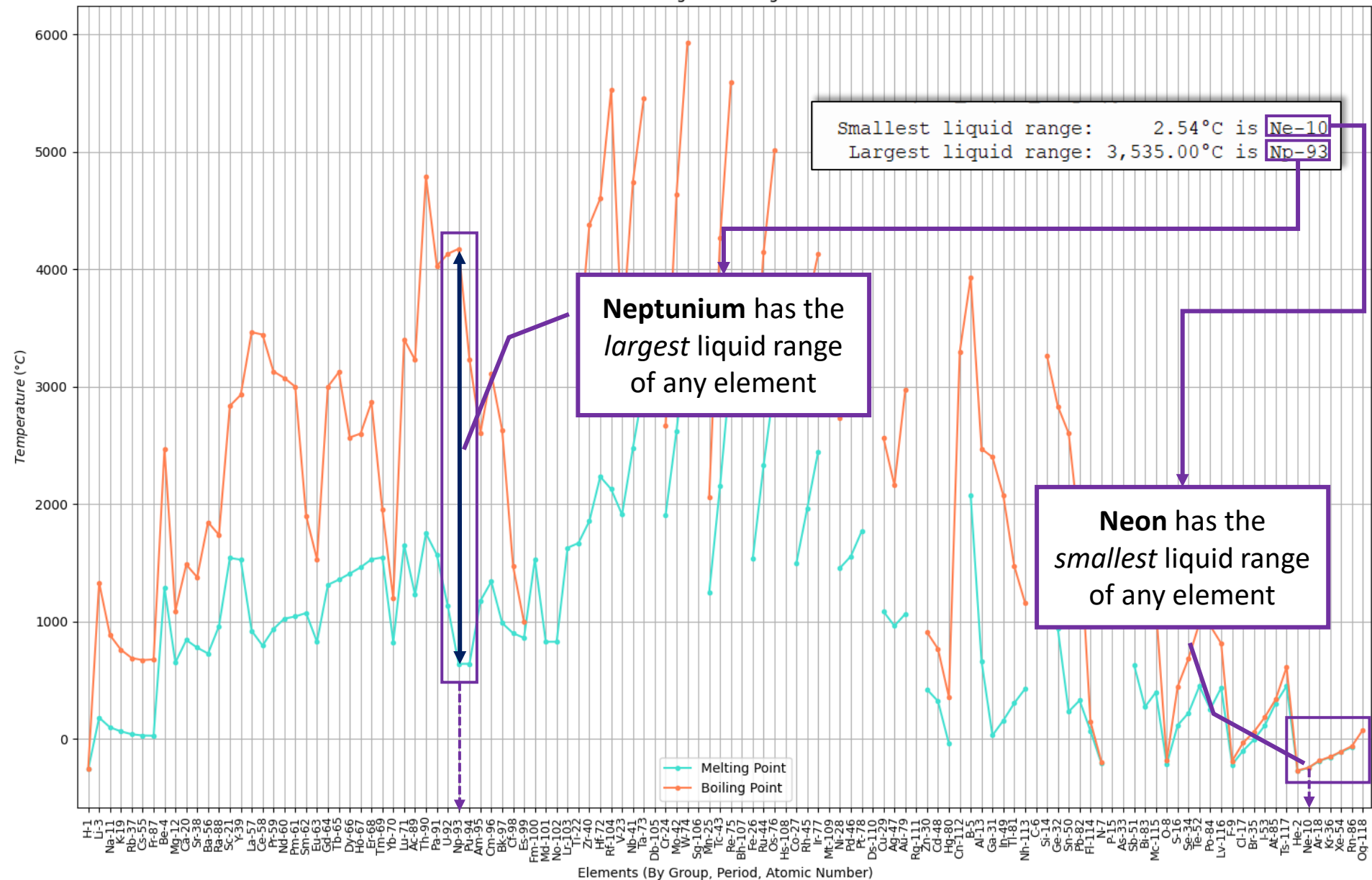
```
↔ Largest liquid range: 3,535.00°C is Np-93
```


Run plot_liquid_range.ipynb – Cell 8

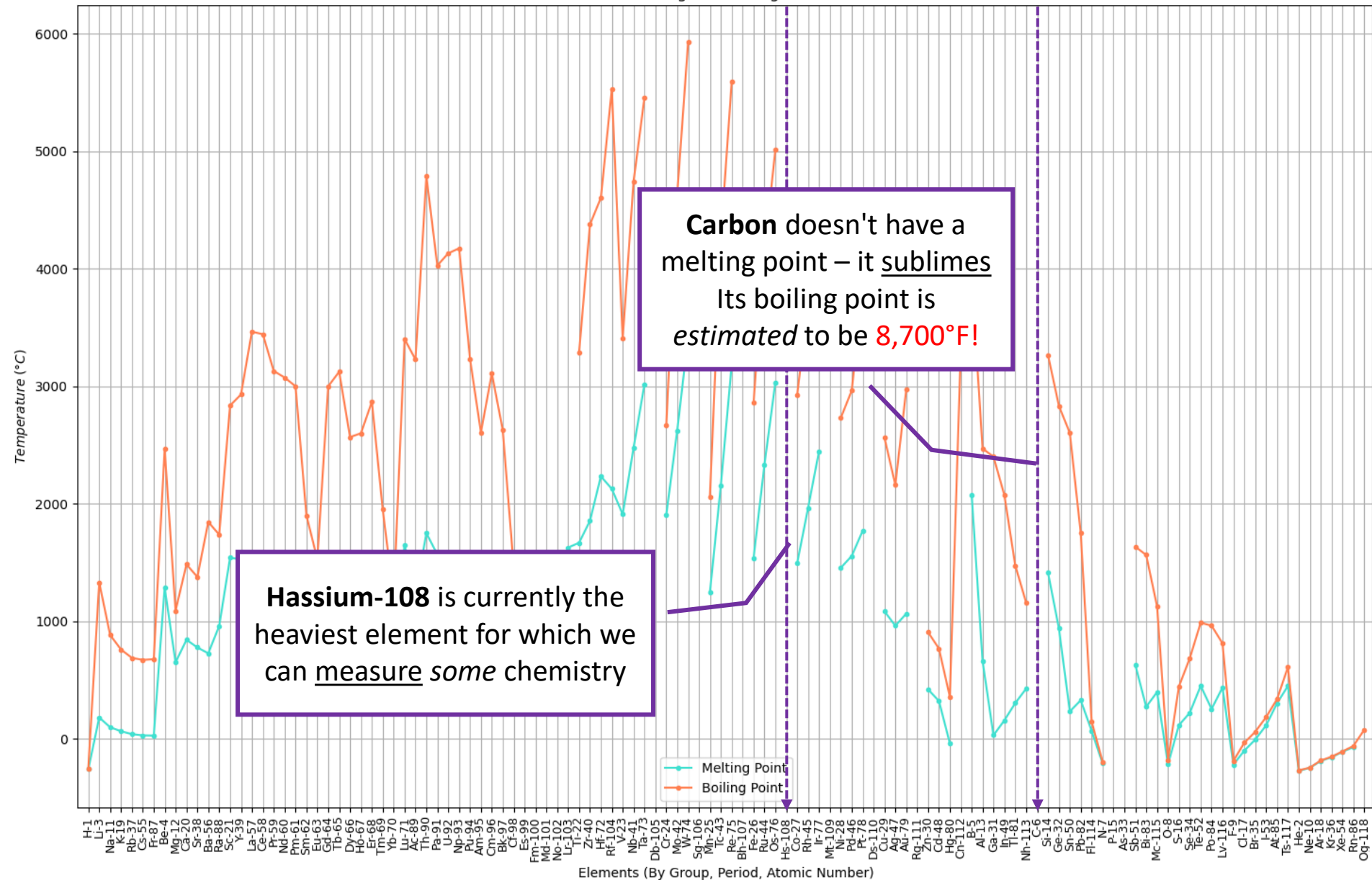
Plot the melting point and boiling point (if available) for every element sorted by group, period, atomic number

```
[8] # Cell 8
plt.figure(figsize=(20, 8))
x = np.arange(len(elements))
plt.plot(x, melt, color="turquoise", marker=".", label="Melting Point")
plt.plot(x, boil, color="coral", marker=".", label="Boiling Point") ← ①
plt.title("Melting and Boiling Point")
plt.xlabel("Elements (By Group, Period, Atomic Number)")
plt.ylabel("Temperature (C)")
ax = plt.gca()
ax.set_xticks(x)
ax.set_xticklabels(data[:, 0], fontsize=9, rotation=90) ← ②
ax.legend(loc="lower center")
ax.grid("on")
plt.show()
```

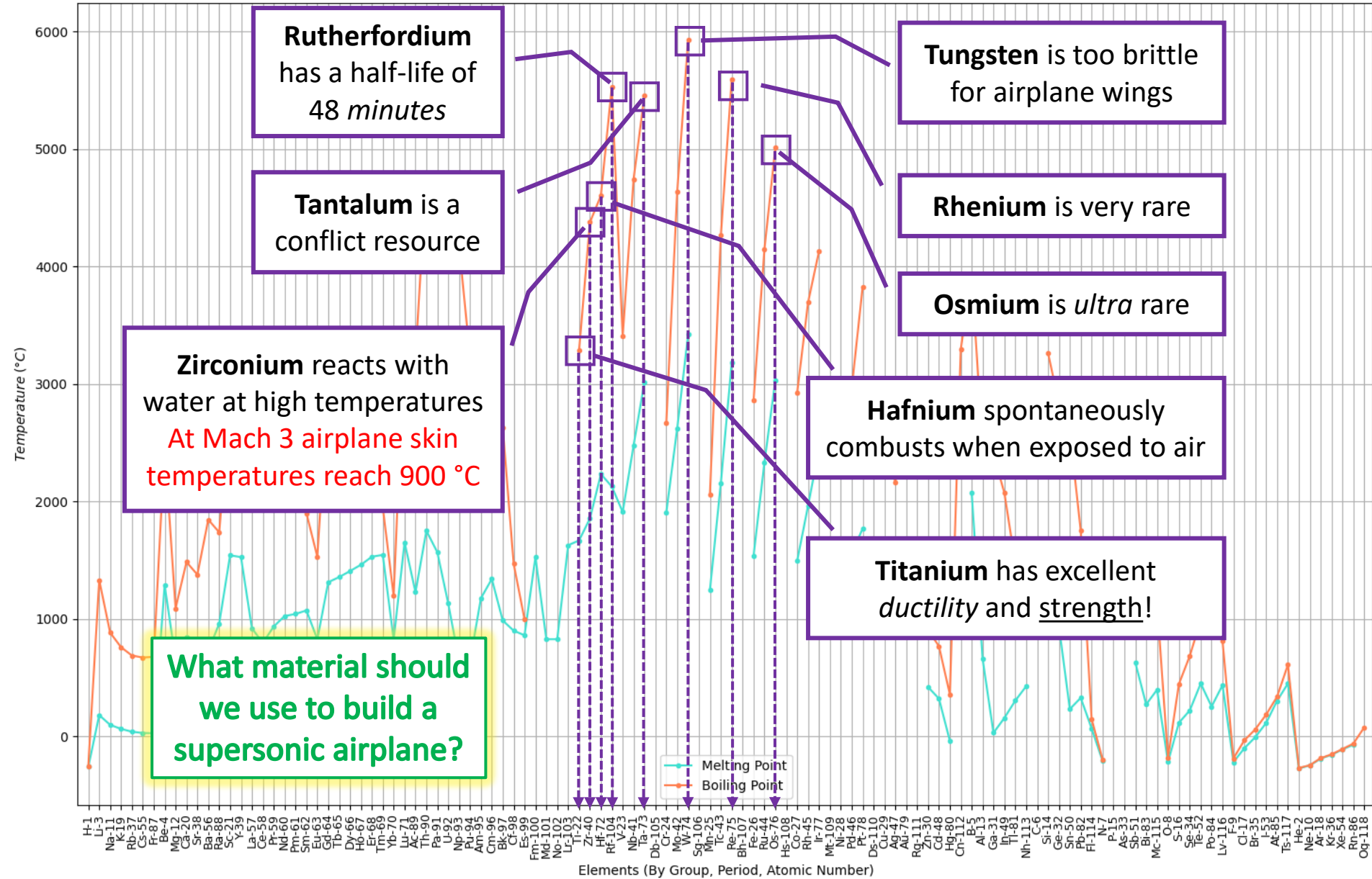
Melting and Boiling Point



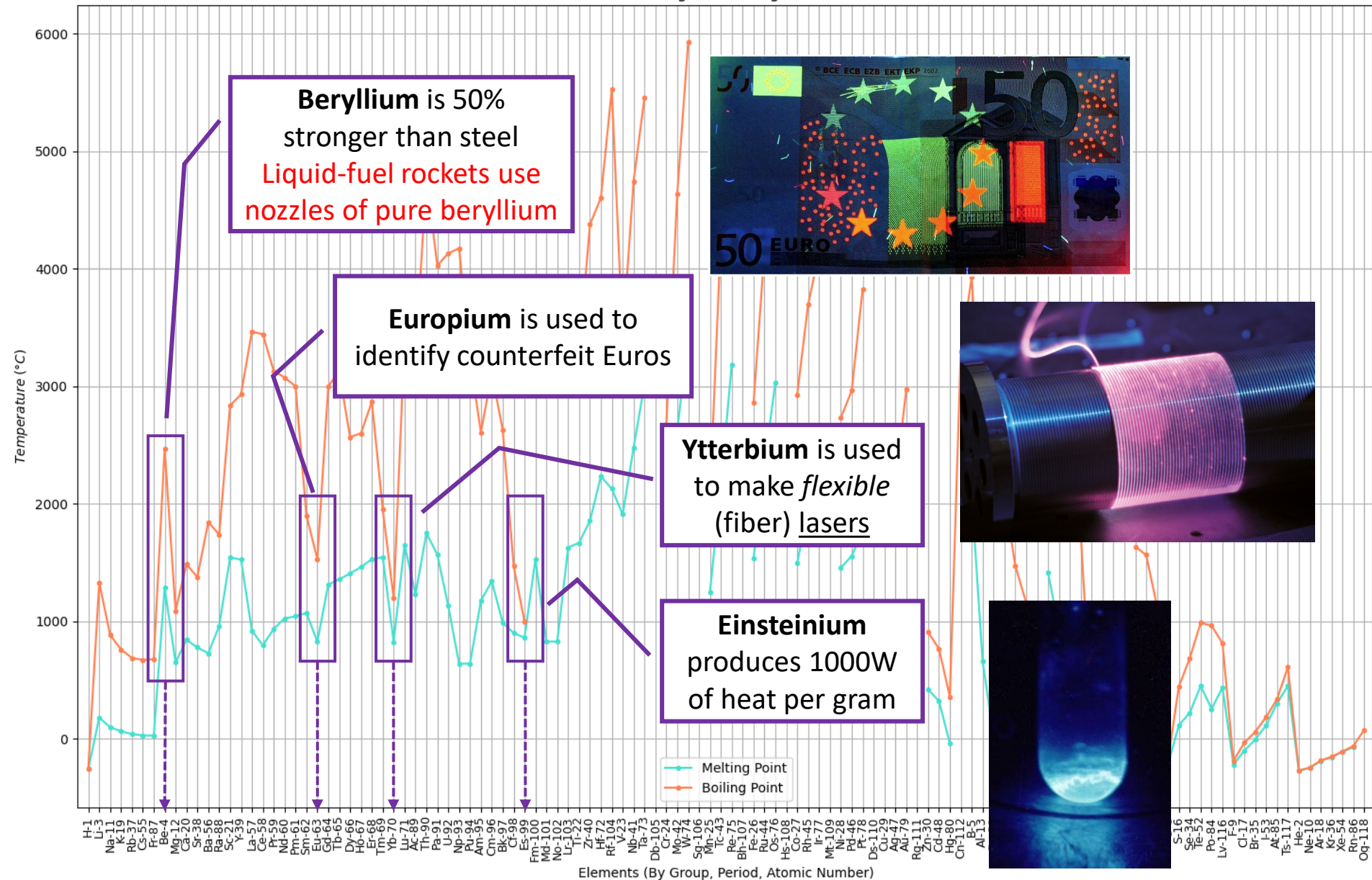
Melting and Boiling Point



Melting and Boiling Point



Melting and Boiling Point




DATA.GOV

<https://catalog.data.gov>

The screenshot displays the DATA.GOV website interface. At the top, the logo and navigation menu (DATA, TOPICS, RESOURCES, STRATEGY, DEVELOPERS, CONTACT) are visible. A search bar contains the word "Biology", which is highlighted with a red box. To the right of the search bar, the "Order by:" dropdown is set to "Popular". Below the search bar, a "Filter by location" section includes a "Clear" button and a text input field for location. A world map is shown below the input field. A red arrow points from the search bar to a red-bordered box containing the text "25,753 datasets found for 'Biology'". The main content area lists several datasets, each with a "Federal" label on the right. The first dataset is "USGS Water-Quality Data for the Nation - National Water Information System (NWIS)" with 125 recent views. The second is "National Database for Autism Research (NDAR)" with 94 recent views. The third is "USGS Surface-Water Data for the Nation - National Water Information System (NWIS)" with 60 recent views. The fourth is "National Database for Clinical Trials Related to Mental Illness (NDCT)" with 56 recent views. On the left side, there are sections for "Topics" (Local Government: 52, Climate: 26) and "Topic Categories" (Arctic: 15, Arctic Ocean, Sea...: 11, Arctic Peoples and...: 10, Ecosystem Vulnerability: 10, Water: 6).

Amazon Open Data Sets

<https://registry.opendata.aws>



Registry of Open Data on AWS

About

This registry exists to help people discover and share datasets that are available via AWS resources. See [recent additions](#) and [learn more about sharing data on AWS](#).

Get started using data quickly by viewing [all tutorials with associated SageMaker Studio Lab notebooks](#).

See [all usage examples for datasets listed in this registry](#).

See datasets from [Allen Institute for Artificial Intelligence \(AI2\)](#), [Digital Earth Africa](#), [Data for Good at Meta](#), [NASA Space Act Agreement](#), [NIH STRIDES](#), [NOAA Open Data Dissemination Program](#), [Space Telescope Science Institute](#), and [Amazon Sustainability Data Initiative](#).

Search datasets (currently 392 matching datasets)

Add to this registry

If you want to add a dataset or example of how to use a dataset to this registry, please follow the instructions on the [Registry of Open Data on AWS GitHub repository](#).

The Cancer Genome Atlas

[cancer](#) [genomic](#) [life sciences](#) [STRIDES](#) [whole genome sequencing](#)

The Cancer Genome Atlas (TCGA), a collaboration between the National Cancer Institute (NCI) and National Human Genome Research Institute (NHGRI), aims to generate comprehensive, multi-dimensional maps of the key genomic changes in major types and subtypes of cancer. TCGA has analyzed matched tumor and normal tissues from 11,000 patients, allowing for the comprehensive characterization of 33 cancer types and subtypes, including 10 rare cancers. The dataset contains open Clinical Supplement, Biospecimen Supplement, RNA-Seq Gene Expression Quantification, miRNA-Seq Isoform Expression Quantificati...

[Details →](#)

Usage examples

- [Comprehensive Analysis of Alternative Splicing Across Tumors from 8,705 Patients](#) by André Kahles, Kjong-Van Lehmann, et al.
- [Broad Institute FireCloud](#) by The Broad Institute of MIT & Harvard
- [The Immune Landscape of Cancer](#) by Vésteinn Thorsson, David L. Gibbs, et al.
- [Genomic, Pathway Network, and Immunologic Features Distinguishing Squamous Carcinomas](#) by Joshua D. Campbell, Christina Yau, et al.
- [Integrated Genomic Analysis of the Ubiquitin Pathway across Cancer Types](#) by Zhongqi Ge, Jake S. Leighton, et al.

[See 29 usage examples →](#)

Kaggle

<https://www.kaggle.com/datasets>

The screenshot displays the Kaggle Datasets interface. On the left is a navigation sidebar with the Kaggle logo and menu items: Create, Home, Competitions, Datasets (highlighted), Code, Discussions, Learn, More, Your Work, and a Recently Viewed section containing 'Yelp Dataset' and 'Iris Dataset (JSON Ver...'. The main content area is titled 'Datasets' and features a search bar with 'JSON' entered. Below the search bar are filter buttons: 'All datasets X', 'Computer Science', 'Computer Vision', 'NLP', and 'Data Visualization'. A red box highlights the text '16,725 Datasets' next to a folder icon. The search results list four datasets:

- Iris Dataset (JSON Version)**
Rachael Tatman · Updated 5 years ago
Usability 7.5 · 1 File (JSON) · 1 kB
- Yelp Dataset**
Yelp, Inc. · Updated 10 months ago
Usability 7.5 · 6 Files (JSON, other) · 4 GB
- Wikidata jsons**
Timo Bozsolik · Updated 3 years ago
Usability 8.8 · 15 Files (JSON, other) · 900 MB
- Drake Lyrics**
Juico Bowley · Updated 2 years ago
Usability 10.0 · 3 Files (JSON, CSV, other) · 782 kB

Identify an Unknown Element

Your lab assistant distilled **50 g** of a gas into a cylinder, but he left without writing down what kind of gas it is. The cylinder has a pressure regulator that adjusts a piston to keep the **pressure at a constant 2.00 atm.** To identify the gas, you measure the cylinder volume at several different temperatures, acquiring the data shown at the right. **What is the gas?**

T ($^{\circ}\text{C}$)	V (L)
-50	11.6
0	14.0
50	16.2
100	19.4
150	21.8

$$PV = nRT$$

SI Units

constant

P = pressure

pascals (Pa)

V = volume

meters³

constant

n = amount of substance

constant

R = ideal gas constant

T = temperature

kelvin (K)

$$V = \left(\frac{nR}{P} \right) T$$

$y = mx$

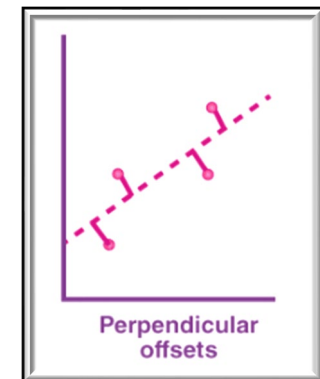
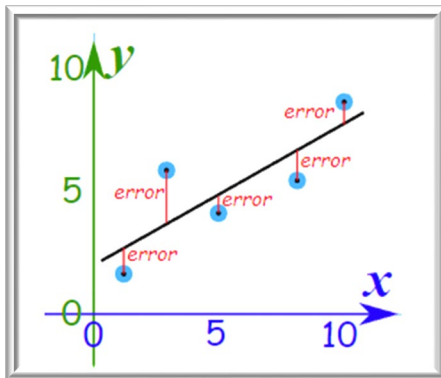
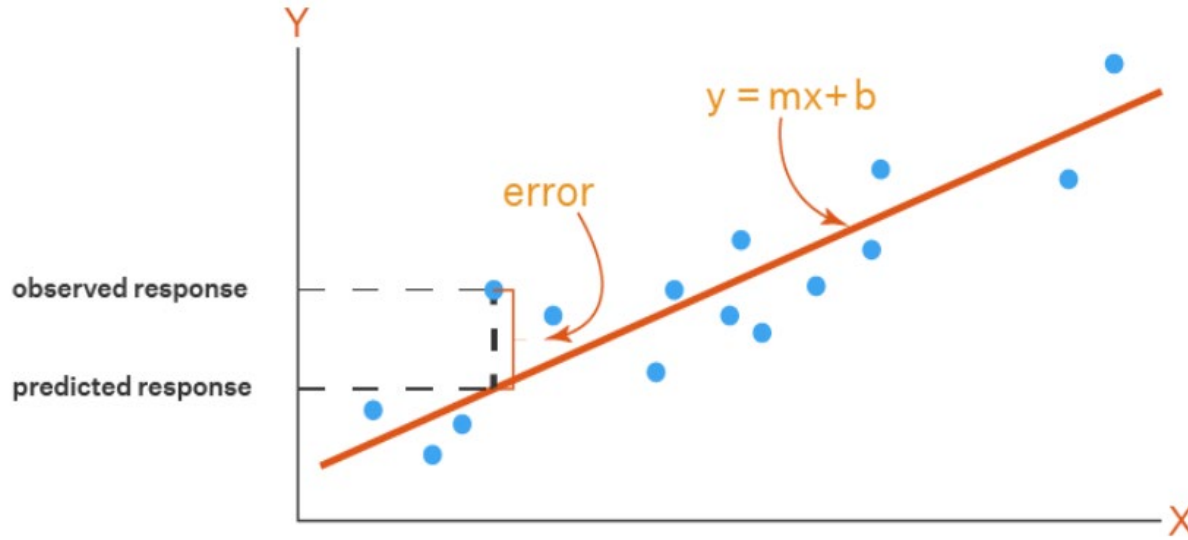
$$V = (m) T$$

constant



Carl Friedrich
Gauss
(1777-1855)

Method of Least Squares



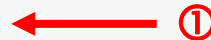
Run identify_element.ipynb – Cells 1..2

Import necessary packages/modules

```
[1] # Cell 1
    from pathlib import Path

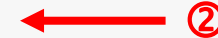
    import matplotlib.pyplot as plt
    import numpy as np
    from google.colab import drive

    import pandas as pd
```



Connect this notebook to your Google Drive

```
[2] # Cell 2
    drive.mount("/content/gdrive", force_remount=True)
    notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
    notebook_path = notebook_path / Path("Session 03 - Computing in Science")
    notebook_path
```



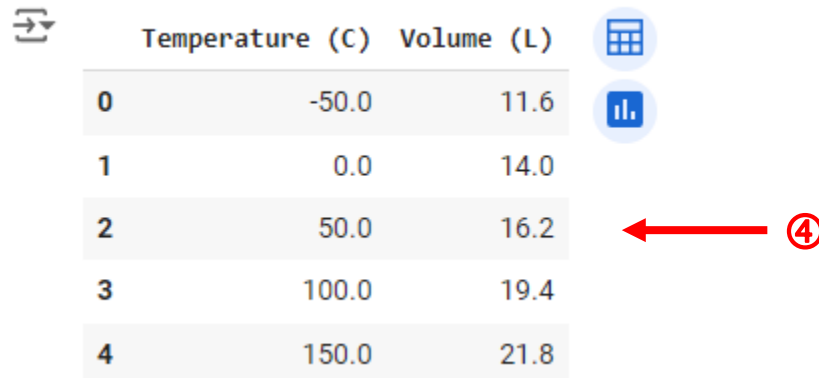
```
↳ Mounted at /content/gdrive
   PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')
```

Run identify_element.ipynb – Cell 3

Generate a numpy array from a CSV (comma separated value) formatted text file

1. The 1st column (index 0) is the temperature in ° Celsius
2. The 2nd column (index 1) is the volume in Liters

```
[3] # Cell 3
file_name = "gas.csv" ← ①
file_path = notebook_path/ file_name
data = np.genfromtxt(file_path, delimiter=",") ← ②
pd.DataFrame(data, columns=["Temperature (C)", "Volume (L)"]) ← ③
```



	Temperature (C)	Volume (L)
0	-50.0	11.6
1	0.0	14.0
2	50.0	16.2
3	100.0	19.4
4	150.0	21.8

④

Run identify_element.ipynb – Cell 4

Convert the experiment data to SI units

1. Temperature should be in ° Kelvin
2. Volume should be in cubic Meters

```
[4] # Cell 4
temperature = data[:, 0] + 273.15 # 1st column to kelvin
volume = data[:, 1] / 1000 # 2nd column to meters cubed
pd.DataFrame({
    "Temperature (K)": temperature,
    "Volume (m^3)": volume
})
```

	Temperature (K)	Volume (m ³)
0	223.15	0.0116
1	273.15	0.0140
2	323.15	0.0162
3	373.15	0.0194
4	423.15	0.0218

Run identify_element.ipynb – Cell 5

Define a function to calculate the line of best fit through the points (x, y)

1. Use Gauss's Linear Regression formulas that minimize the error ← ①
2. The (x) array holds the independent variable values
3. The (y) array holds the dependent variable values
4. The function returns m (slope) and b (y-intercept) of the line of best fit passing through the (x, y) points

```
[5] # Cell 5
def fit_linear(x, y): ← ②
    m = len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)
    m = m / (len(x) * np.sum(x**2) - np.sum(x) ** 2)
    b = (np.sum(y) - m * np.sum(x)) / len(x)
    return m, b ← ③

# Calculate line of best fit
slope, yint = fit_linear(temperature, volume) ← ④
print(f"slope: {slope:.8f}")
print(f"y-intercept: {yint:.8f}")
```

$$m = \frac{n\sum(XY) - \sum Y\sum X}{n\sum(X^2) - (\sum X)^2}$$

$$b = \frac{\sum Y - m\sum X}{n}$$

```
↔ slope: 0.00005160 ← ⑤
   y-intercept: -0.00007454
```

Slope of line
of best fit $m = \frac{\Delta y}{\Delta x} = \frac{V}{T}$

Run identify_element.ipynb – Cell 6..7

Leverage the ideal gas law $PV = nRT$ to solve for n (number of moles of the unknown gas)

```
[6] # Cell 6
    p = 2.0 * 101_325 # Convert 2.0 atm (given) to pascals ← ①
    r = 8.31446261815324 # Gas constant (SI units)
    n = p / r * slope # Moles of gas (rearrange ideal gas law equation) ← ②
    print(f"Number of moles of gas: {n:.8f}")
```

⇒ Number of moles of gas: 1.25765675 $PV = nRT$ $n = \left(\frac{P}{R}\right) \left(\frac{V}{T}\right) = \frac{P}{R} (m)$

Calculate the atomic mass of the homogenous (pure) gas in the cylinder

```
[7] # Cell 7
    m_sample = 50 # (given) grams
    atomic_mass = m_sample / n # sample mass divided by number of moles ← ③
    print(f"Atomic mass of unknown gas: {atomic_mass:8f}u")
```

⇒ Atomic mass of unknown gas: 39.756476u ← ④

Molar Mass = Atomic Mass (specific isotope) \approx Atomic Weight (avg)

Run identify_element.ipynb – Cell 8

In preparation for plotting the Temperature vs. Volume line for this gas:

1. Create a linear space of temperature (t) values spanning 0 to 500° Kelvin
2. Display the numpy array (t)

```
[8] # Cell 8  
t = np.linspace(0, 500)  
display(t)
```

```
↳ array([ 0.          , 10.20408163, 20.40816327, 30.6122449 ,  
         40.81632653, 51.02040816, 61.2244898 , 71.42857143,  
         81.63265306, 91.83673469, 102.04081633, 112.24489796,  
        122.44897959, 132.65306122, 142.85714286, 153.06122449,  
        163.26530612, 173.46938776, 183.67346939, 193.87755102,  
        204.08163265, 214.28571429, 224.48979592, 234.69387755,  
        244.89795918, 255.10204082, 265.30612245, 275.51020408,  
        285.71428571, 295.91836735, 306.12244898, 316.32653061,  
        326.53061224, 336.73469388, 346.93877551, 357.14285714,  
        367.34693878, 377.55102041, 387.75510204, 397.95918367,  
        408.16326531, 418.36734694, 428.57142857, 438.7755102 ,  
        448.97959184, 459.18367347, 469.3877551 , 479.59183673,  
        489.79591837, 500.          ])
```

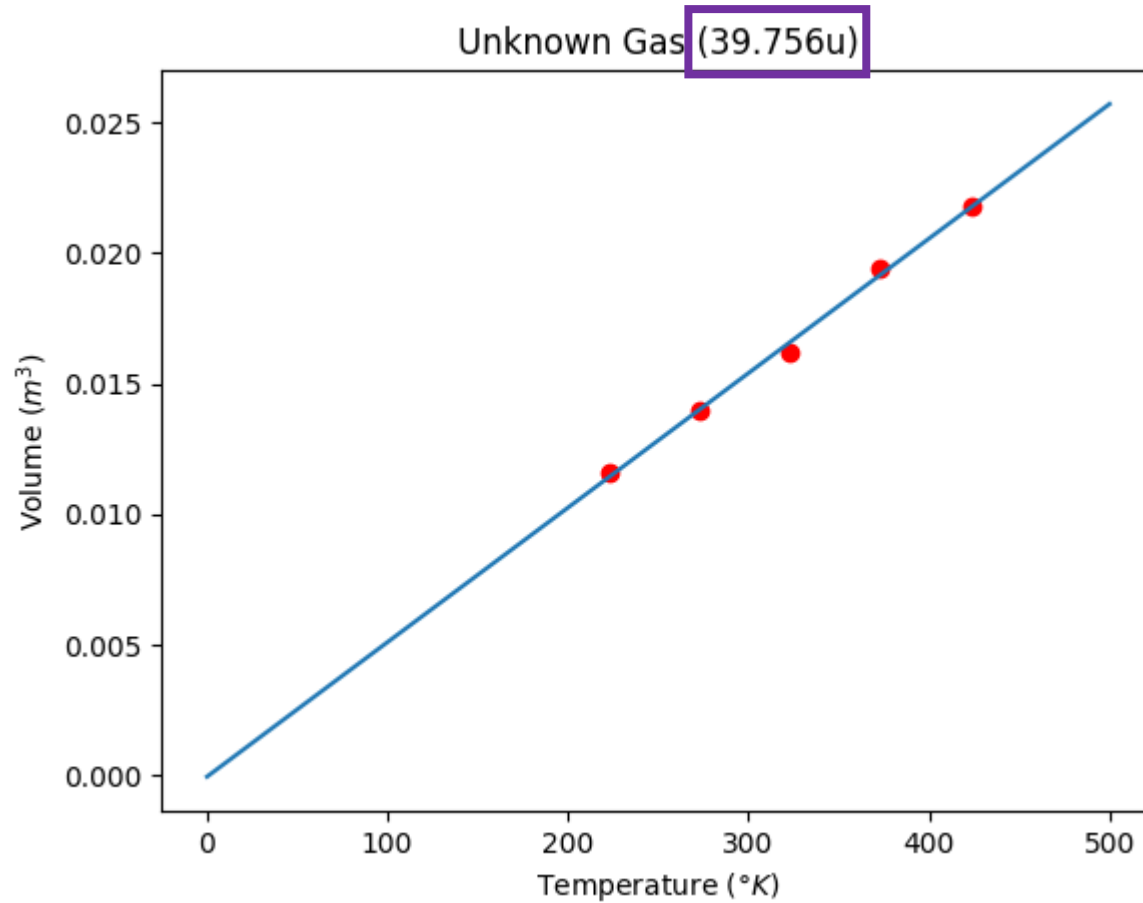

Run identify_element.ipynb – Cell 9

Plot the Temperature vs. Volume line for this gas ← ①

1. Superimpose the data points from the CSV file
2. Display the atomic mass of the unknown element in the graph's title

```
[9] # Cell 9
plt.scatter(temperature, volume, color="red") ← ②
plt.plot(t, slope * t + yint) ← ③
plt.title(f"Unknown Gas ({atomic_mass:.3f}u)")
plt.xlabel(r"Temperature  $(\text{degree K})$ ")
plt.ylabel("Volume ( $\text{m}^3$ )")
plt.show()
```

View identify_element.ipynb – Cell 9



Unknown Gas (39.756u)

- period group 1*
- Alkali
 - Alkali
 - Trans
 - Other
 - Other

1	H [1.00784, 1.00811]			
2	Li [6.938, 6.997]	Be 9.0121831		
3	Na 22.98976928	Mg [24.304, 24.307]		
4	K 39.0983	Ca 40.078	Sc 44.955908	Ti 47.867
5	Rb 85.4678	Sr 87.62	Y 88.90584	Zr 91.224
6	Cs 132.905452	Ba 137.327	La 138.90547	Hf 178.486
7	Fr (223)	Ra (226)	Ac (227)	Rf (261)

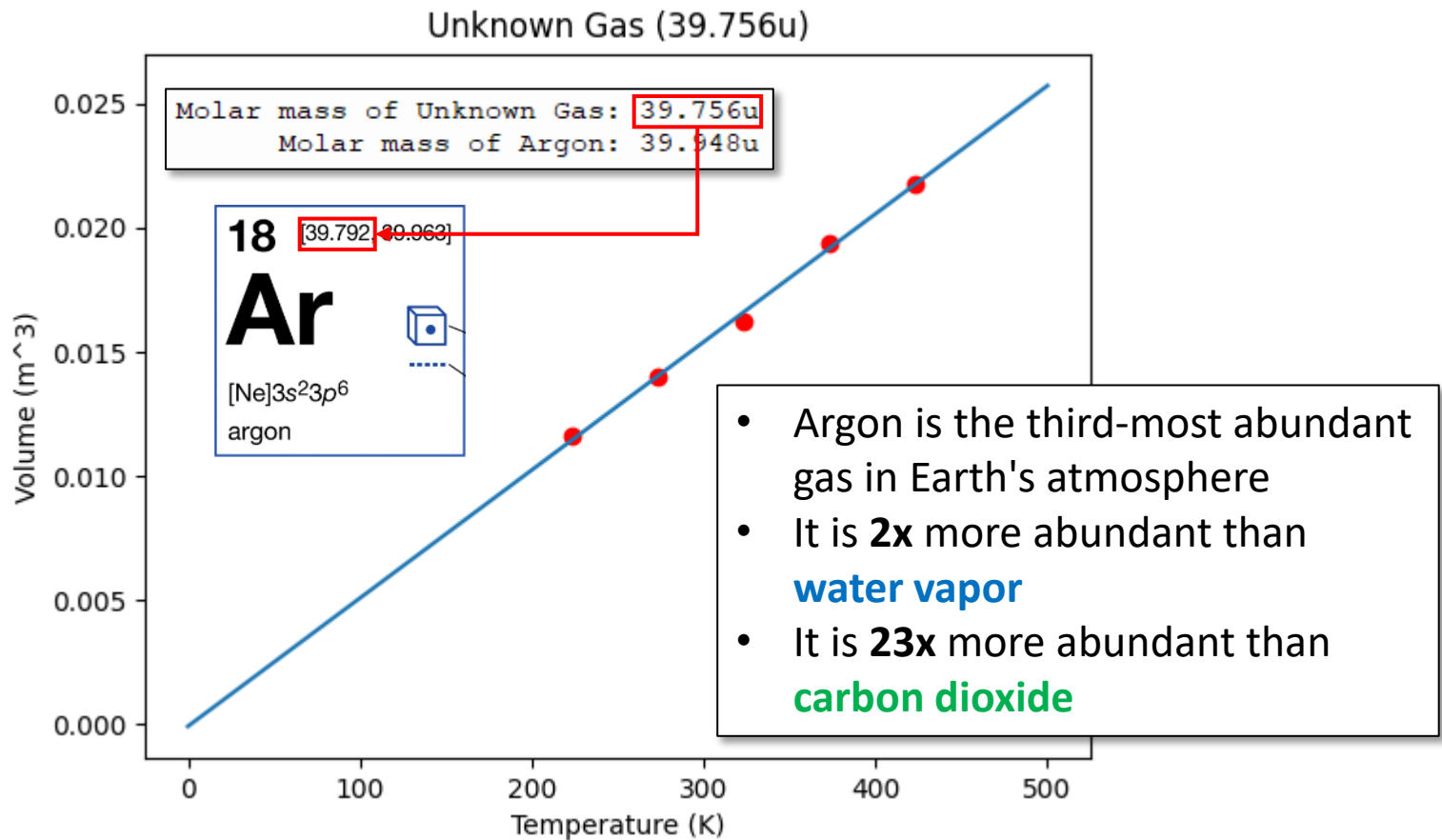
16	17	2
		He 4.002602
8 O [15.99903, 15.99977]	9 F 18.998403163	10 Ne 20.1797
16 S [32.059, 32.076]	17 Cl [35.446, 35.457]	18 Ar [39.792, 39.963]
34 Se 78.971	35 Br [79.901, 79.907]	36 Kr 83.798

			18
			2 He 4.002602
7 N [14.00643, 14.00728]	8 O [15.99903, 15.99977]	9 F 18.998403163	10 Ne 20.1797
15 P [30.973762]	16 S [32.059, 32.076]	17 Cl [35.446, 35.457]	18 Ar [39.792, 39.963]
33 As 74.921595	34 Se 78.971	35 Br [79.901, 79.907]	36 Kr 83.798
51 Sb 121.76	52 Te 127.6	53 I 126.90447	54 Xe 131.293
83 Bi 208.9804	84 Po (209)	85 At (210)	86 Rn (222)
115 Mc (288)	116 Lv (293)	117 Ts (294)	118 Og (294)

69 Tm 168.934218	70 Yb 173.045	71 Lu 174.9668
101 Md (258)	102 No (259)	103 Lr (262)



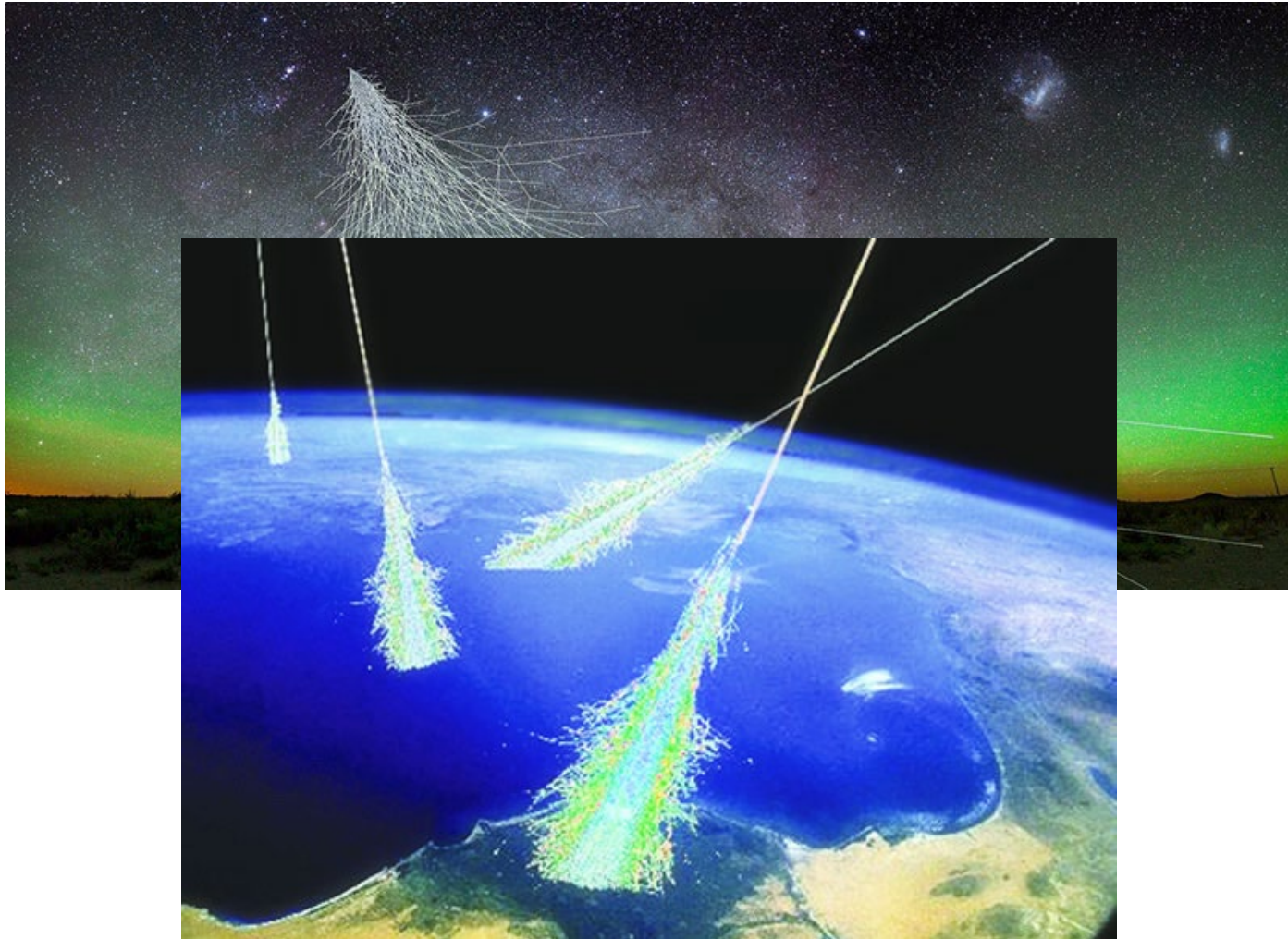
Check identify_element.ipynb – Cell 9



Cosmic Rays

- Cosmic rays entering the Earth's atmosphere collide with gas molecules, creating secondary particles
 - Your scientist has developed an instrument to capture the trajectory of these secondary particles as they rain down

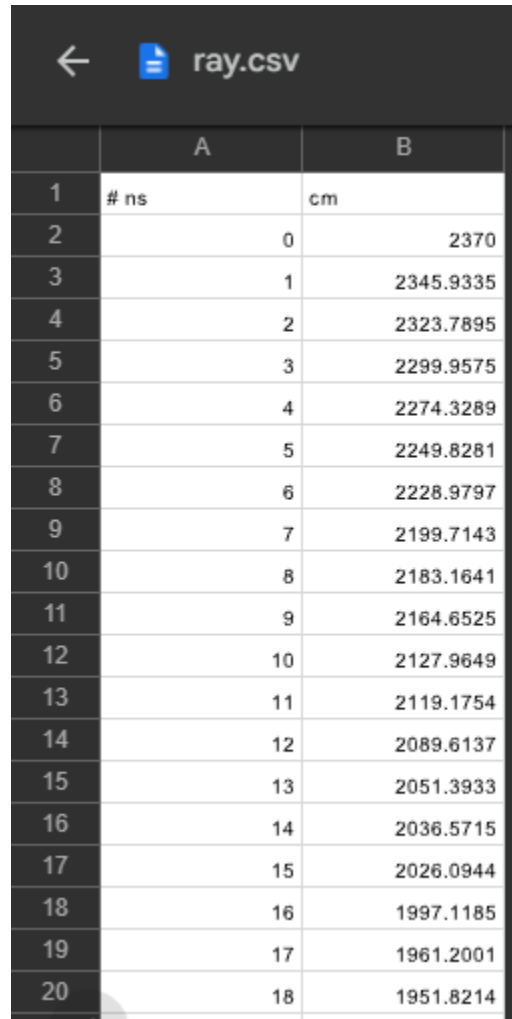
Cosmic Rays



Cosmic Rays

- Cosmic rays entering the Earth's atmosphere collide with gas molecules, creating secondary particles
 - Your scientist has developed an instrument to capture the trajectory of these secondary particles as they rain down
 - He has given you a data file (**ray.csv**) of a particle's height (in **centimeters**) over the final **nanoseconds** before its impact

Cosmic Rays

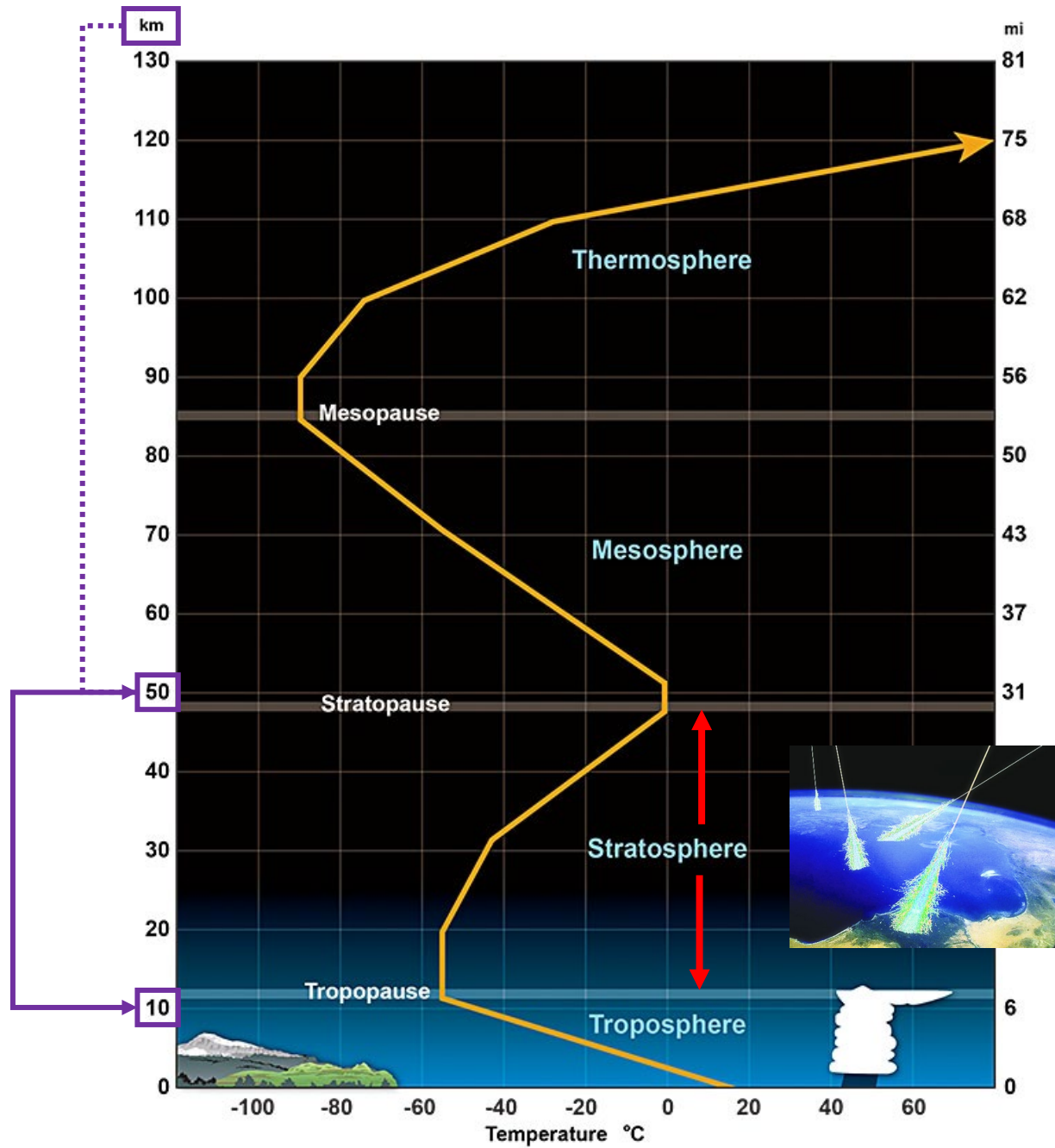


A screenshot of a mobile spreadsheet application displaying a CSV file named "ray.csv". The spreadsheet has two columns, A and B, and 20 rows of data. The first row contains headers "# ns" for column A and "cm" for column B. The data shows a decreasing trend in column B as the value in column A increases from 0 to 18.

	A	B
1	# ns	cm
2	0	2370
3	1	2345.9335
4	2	2323.7895
5	3	2299.9575
6	4	2274.3289
7	5	2249.8281
8	6	2228.9797
9	7	2199.7143
10	8	2183.1641
11	9	2164.6525
12	10	2127.9649
13	11	2119.1754
14	12	2089.6137
15	13	2051.3933
16	14	2036.5715
17	15	2026.0944
18	16	1997.1185
19	17	1961.2001
20	18	1951.8214

Cosmic Rays

- Cosmic rays entering the Earth's atmosphere collide with gas molecules, creating secondary particles
 - Your scientist has developed an instrument to capture the trajectory of these secondary particles as they rain down
 - He has given you a data file (**ray.csv**) of a particle's height (in **centimeters**) over the final **nanoseconds** before its impact
 - The scientist knows the secondary particle was **not** accelerating and lived for only **0.1743 milliseconds**
 - He wants you to determine its velocity (relative to **c**) and the height (in **km**) in the *stratosphere* at which it was originally emitted
- How would you display the particle's path and **use the line of best fit** to determine those *two* unknowns?



Run cosmic_rays.ipynb – Cells 1...2

Import necessary packages/modules

```
[1] # Cell 1
    from pathlib import Path

    import matplotlib.pyplot as plt
    import numpy as np
    from google.colab import drive ← ①

    import pandas as pd
```

Connect this notebook to your Google Drive

```
[2] # Cell 2
    drive.mount("/content/gdrive", force_remount=True) ← ②
    notebook_path = Path("/content/gdrive/MyDrive/asp-hs")
    notebook_path = notebook_path / Path("Session 03 - Computing in Science")
    notebook_path
```

Mounted at /content/gdrive
PosixPath('/content/gdrive/MyDrive/asp-hs/Session 03 - Computing in Science')

Run cosmic_rays.ipynb – Cell 3

Generate a numpy array from a CSV (comma separated value) formatted text file

Print the first 10 rows in the generated array

```
[3] # Cell 3
file_name = "ray.csv"
file_path = notebook_path/ file_name
data = np.genfromtxt(file_path, delimiter=",")
pd.DataFrame(data[:10], columns=["Time (ns)", "Height (cm)"])
```



Time (ns) Height (cm)

0	0.0	2370.0000
1	1.0	2345.9335
2	2.0	2323.7895
3	3.0	2299.9575
4	4.0	2274.3289
5	5.0	2249.8281
6	6.0	2228.9797
7	7.0	2199.7143
8	8.0	2183.1641
9	9.0	2164.6525



Run cosmic_rays.ipynb – Cell 4

Slice the data array by columns into two 1D arrays ← ①

1. The 1st column (index 0) is the time (t) in seconds - the *independent* variable ← ②
2. The 2nd column (index 1) is the height (h) above ground level in centimeters - the *dependent* variable
3. Print the first five elements in each array ← ③

```
[4] # Cell 4
t = data[:, 0] ← ④
h = data[:, 1]
print(f"{t[:5] = }") ← ⑤
print(f"{h[:5] = }")
```

```
→ t[:5] = array([0., 1., 2., 3., 4.]) ← ⑥
h[:5] = array([2370.      , 2345.9335, 2323.7895, 2299.9575, 2274.3289])
```

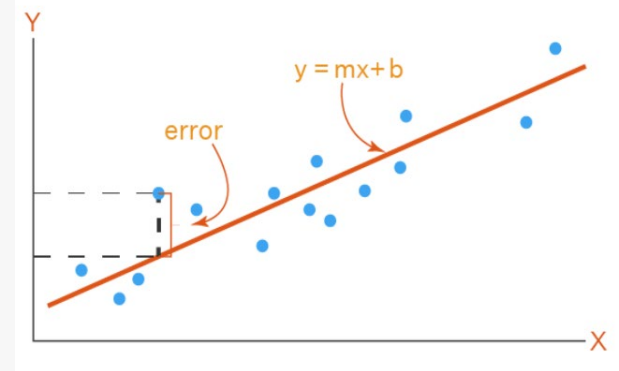
Run cosmic_rays.ipynb – Cell 5

Define a function to calculate the line of best fit through the points (x, y)

1. Use Gauss's Linear Regression formulas that minimize the error ← ①
2. The (x) array holds the independent variable values
3. The (y) array holds the dependent variable values
4. The function returns m (slope) and b (y-intercept) of the line of best fit passing through the (x, y) points

```
[5] # Cell 5
def fit_linear(x, y): ← ②
    m = len(x) * np.sum(x * y) - np.sum(x) * np.sum(y)
    m = m / (len(x) * np.sum(x**2) - np.sum(x) ** 2)
    b = (np.sum(y) - m * np.sum(x)) / len(x)
    return m, b

# Calculate line of best fit
slope, yint = fit_linear(t, h) ← ③
print(f"slope: {slope:.8f}")
print(f"y-intercept: {yint:.8f}")
```



```
⇒ slope: -23.80824990 ← ④
   y-intercept: 2369.97885028
```

Run cosmic_rays.ipynb – Cell 6

Calculate the particle's origination height oh and initial velocity v ← ①

1. The origination height should be in kilometers (km) ← ②
2. The velocity should be relative to the speed of light (c)
3. The radioactive particle existed for only $0.1743\ ms$ before it decayed ← ③
4. The speed of light is $29.98\ cm/ns$ ← ④

```
[6] # Cell 6
oh = (slope * 1e9 / 100) * (0.1743 / 1e3) / 1000
v = slope / 29.98
print(f"Velocity = {v:.2f}c")
print(f"Origination Height = {oh:,.2f} km")
```

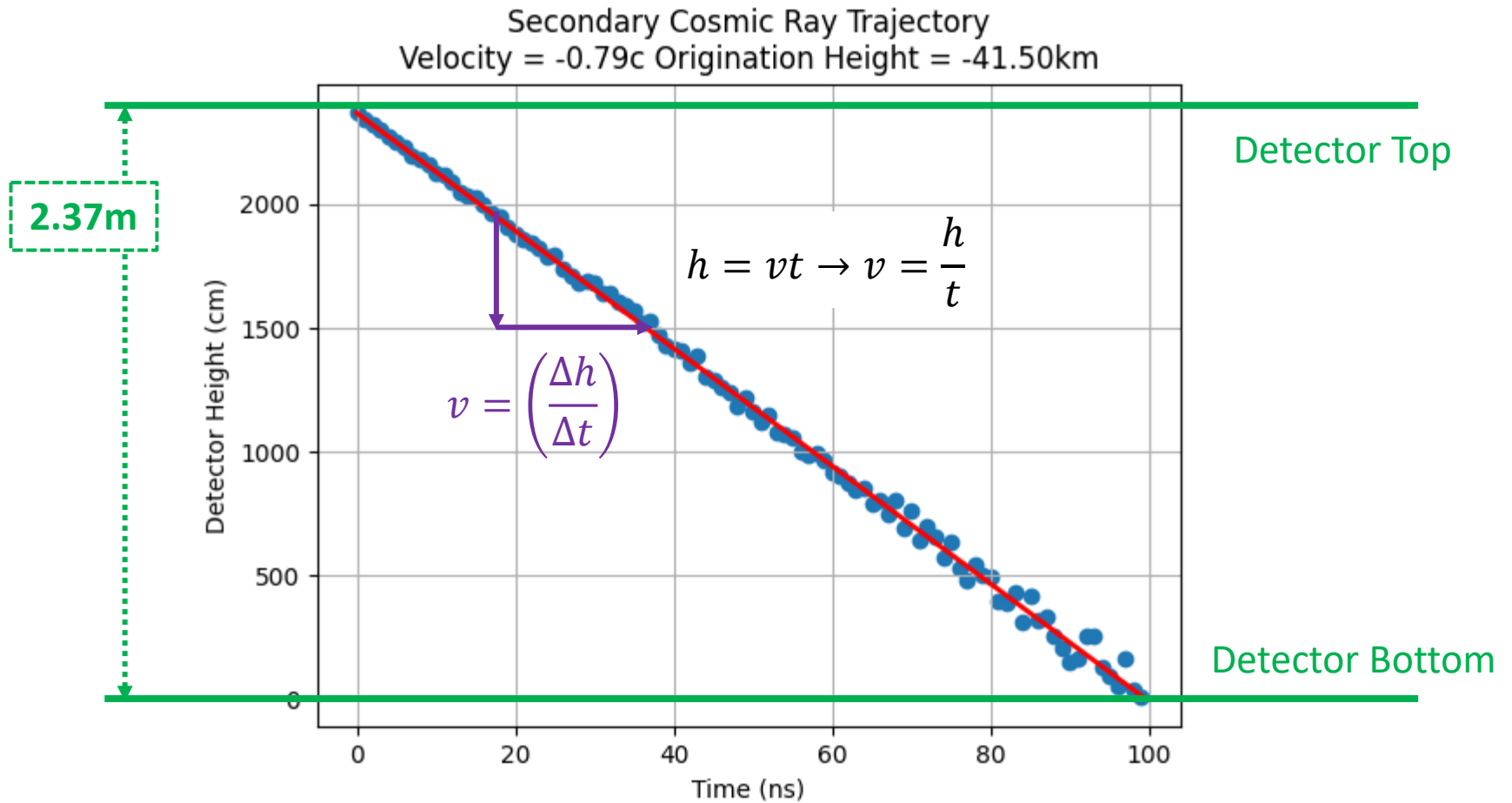
```
↔ Velocity = -0.79c
   Origination Height = -41.50 km ← ⑥
```

Run cosmic_rays.ipynb – Cell 7

Plot the trajectory of the particle in the detector as height (cm) vs. time (ns) ← ①

```
[7] # Cell 7
plt.figure(figsize=(12, 8))
plt.scatter(t, h) ← ②
plt.plot(t, slope * t + yint, color="red", linewidth=2) ← ③
plt.title(
    "Secondary Cosmic Ray Trajectory\n"
    f"Velocity = {v:.2f}c " ← ④
    f"Origination Height = {oh:,.2f}km",
)
plt.xlabel("Time (ns)") ← ⑤
plt.ylabel("Detector Height (cm)")
plt.grid("on")
plt.show()
```


View cosmic_rays.ipynb – Cell 7



Dimensional Analysis

```
# Calculate origination height (oh) and initial velocity (v)
oh = (slope * 1e9 / 100) * (0.1743 / 1e3) / 1000
c = 29.98 # speed of light in cm/ns
v = slope / c
```

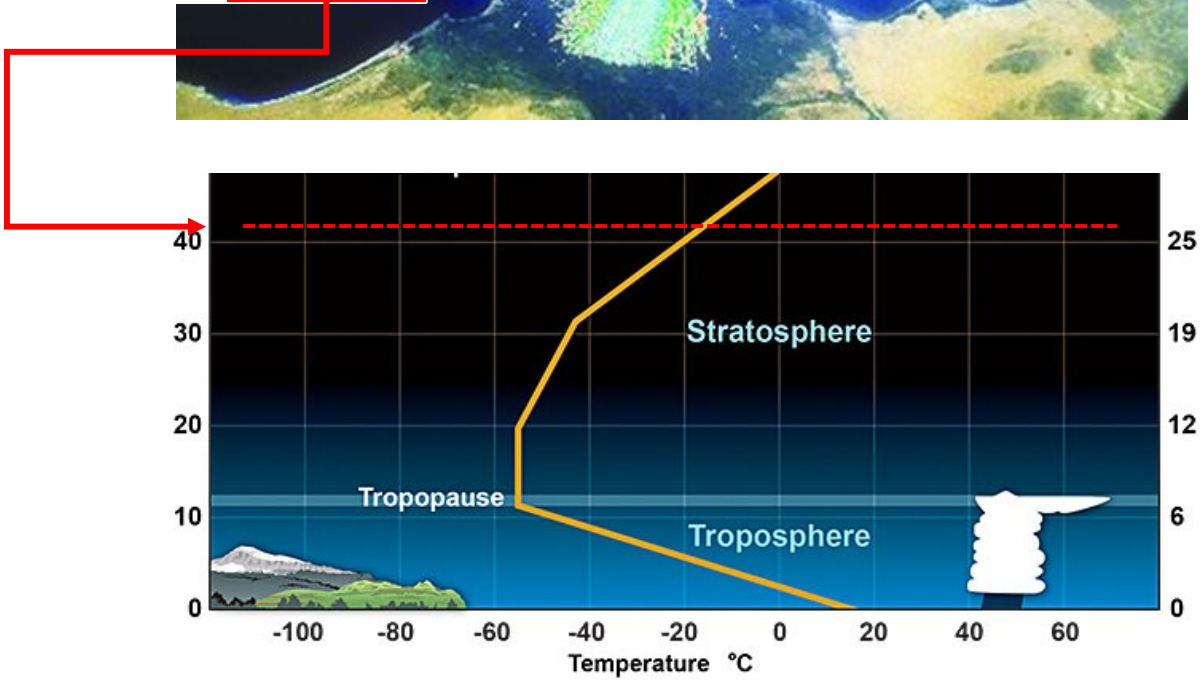
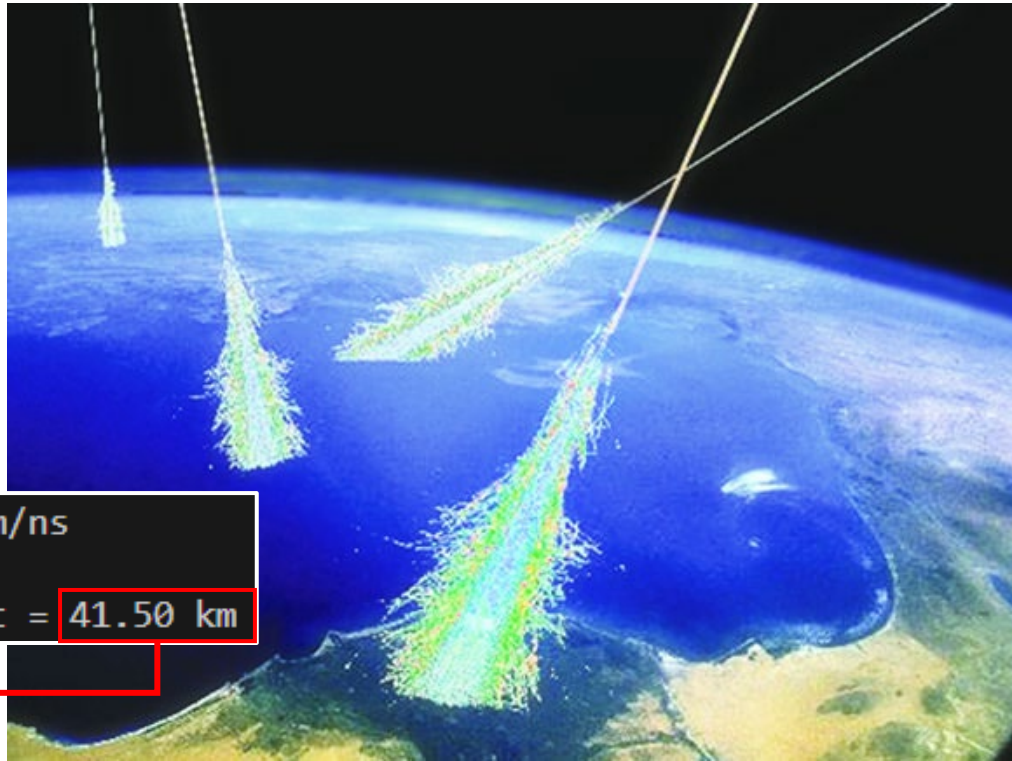
$$h = vt \rightarrow v = \frac{h}{t} \quad \text{Slope} = -23.8082 \text{ cm/ns}$$

$$v = \frac{23.8082 \text{ cm}}{\text{ns}} \times \frac{10^9 \text{ ns}}{1 \text{ s}} \times \frac{1 \text{ m}}{100 \text{ cm}} \times \frac{1 \text{ s}}{299,709,000 \text{ m}} = 0.79 c$$

Given the
lifetime of
the particle

$$t = \frac{0.1743 \text{ ms}}{1} \times \frac{1 \text{ s}}{1000 \text{ ms}} = 0.0001743 \text{ s}$$

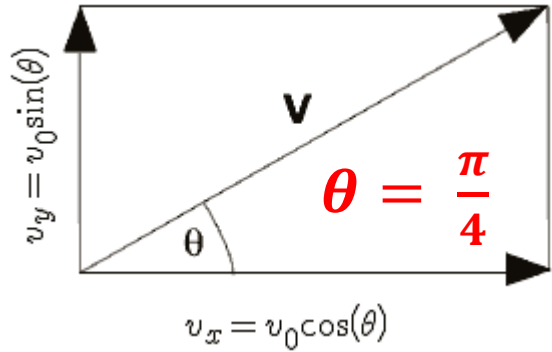
$$h = (238,082,000) \frac{\text{m}}{\text{s}} \times (0.0001743) \frac{\text{s}}{1} \times \frac{1 \text{ km}}{1000 \text{ m}} = 41.50 \text{ km}$$



Projectile Motion



Projectile Motion



v_0 = Initial velocity leaving the cannon

$$x = v_0 * t * \cos(\theta)$$

$$y = v_0 * t * \sin(\theta) - \frac{1}{2} g t^2$$

$$t = \frac{x}{v_0 * \cos(\theta)}$$

$$y = \tan(\theta) * x - \frac{g}{2 * v_0^2 * \cos^2(\theta)} * x^2$$

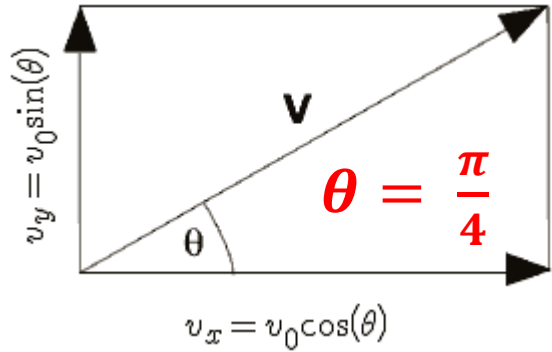
Given the **Range = 30m**,
what does v_0 need to be?

$$h = \frac{v_0^2 \sin^2(\theta)}{2g}$$

$$\text{Range} = \frac{4h}{\tan(\theta)}$$

This is the **equation of motion** that allows us to **plot y as x increases** from launch point to trampoline

Projectile Motion



Given the Range = 30m,
what does v_0 need to be?

$$v_0 = \sqrt{\frac{\text{Range} * g}{\sin 2\theta}}$$

v_0 = Initial velocity leaving the cannon

$$x = v_0 * t * \cos(\theta)$$

$$y = v_0 * t * \sin(\theta) - \frac{1}{2} g t^2$$

$$t = \frac{x}{v_0 * \cos(\theta)}$$

$$y = \tan(\theta) * x - \frac{g}{2 * v_0^2 * \cos^2(\theta)} * x^2$$

This is the **equation of motion** that allows us to **plot y as x increases** from launch point to trampoline

Run projectile_motion.ipynb – Cells 1..2

Import needed packages/modules

```
[1] # Cell 1
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Rectangle ← ①
```

Specify the simulation constants

1. The trampoline is 30m from the cannon
2. The cannon is at a fixed launch angle of 45° ← ②
3. The acceleration due to gravity is $9.81 \frac{m}{s^2}$
4. The initial velocity v_0 is 15 m/s

```
[2] # Cell 2
rng = 30 # m
theta = np.radians(45) # 45 degree launch angle ← ③
g = 9.81 # m/s^2
v0 = 15 # m/s

print(*[{k: v} for k, v in globals().items()
        if not k.startswith(('_', 'In', 'Out')) and not callable(v)], sep='\n') ← ④
```

```
↳ {'plt': <module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
   {'np': <module 'numpy' from '/usr/local/lib/python3.10/dist-packages/numpy/__init__.py'>}
   {'rng': 30}
   {'theta': 0.7853981633974483} ← ⑤
   {'g': 9.81}
   {'v0': 15}
```

Run projectile_motion.ipynb – Cell 3

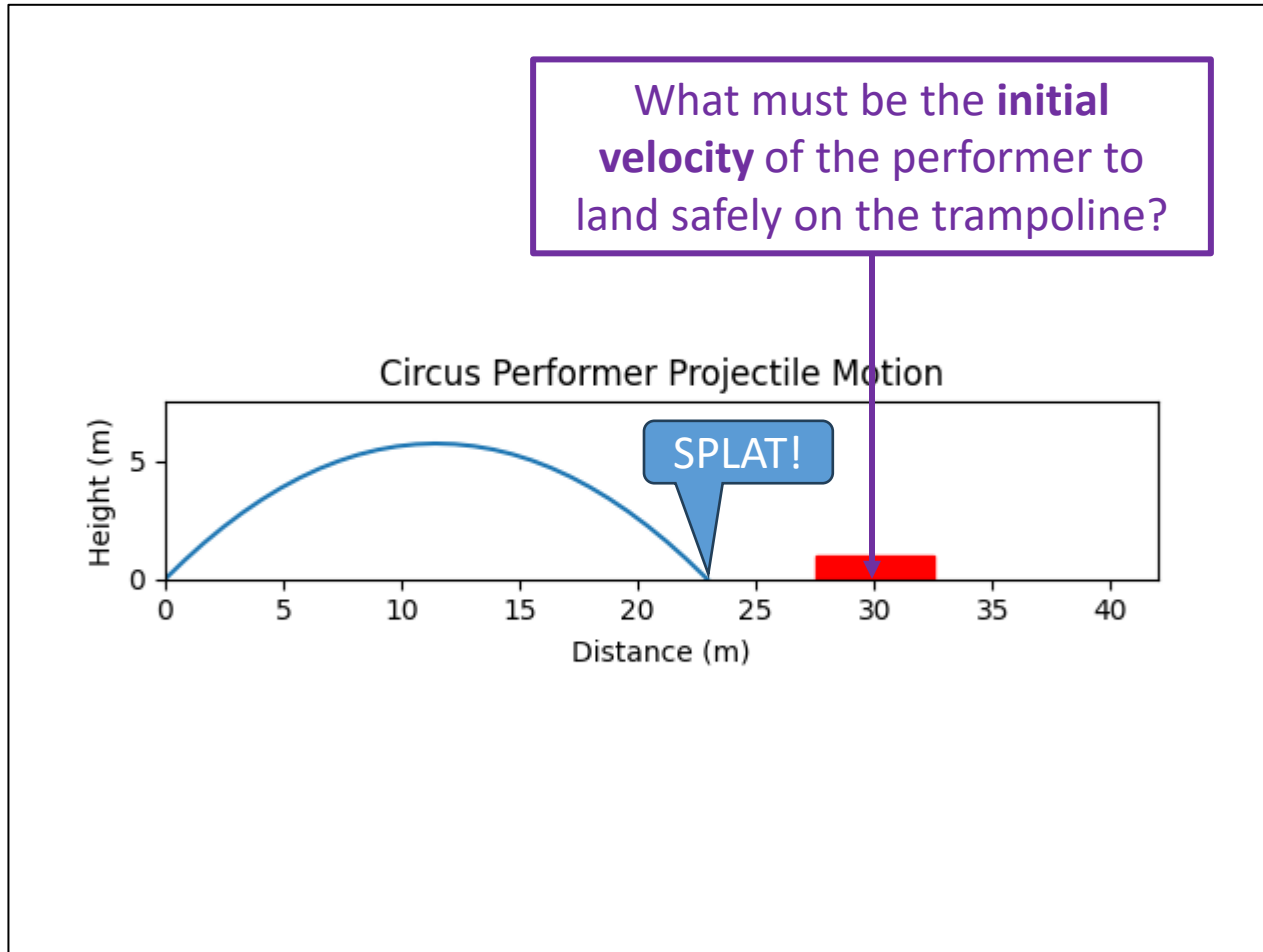
Create a function to calculate and plot the trajectory of the circus performer

- Create a linear space for the independent variable that spans $0 \leq x \leq 40$ ← ①
- $y = \tan(\theta) \times x - \frac{g}{2 \times v_0^2 \times \cos^2(\theta)} \times x^2$ ← ②

```
[3] # Cell 3
def plot_trajectory():
    x = np.linspace(0, 40)
    y = np.tan(theta) * x - (g / (2 * v0**2 * np.cos(theta) ** 2)) * x**2 ← ③
    plt.plot(x, y) ← ④
    plt.title("Circus Performer Projectile Motion")
    plt.xlabel("Distance (m)")
    plt.ylabel("Height (m)")
    plt.xlim(left=0)
    plt.ylim(bottom=0)
    ax = plt.gca()
    ax.add_patch(Rectangle((27.5, 0), 5, 1, color="red")) ← ⑤
    ax.set_aspect("equal")
    plt.show()

plot_trajectory()
```


Check projectile_motion.ipynb – Cell 3



Run projectile_motion.ipynb – Cell 4

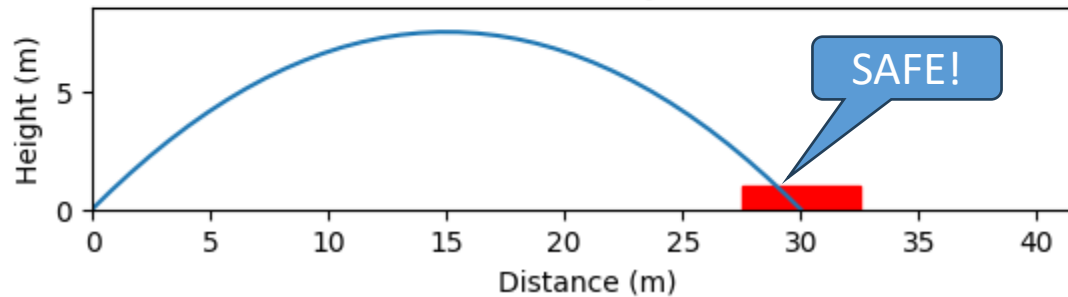
Set the correct initial launch velocity v_0 and recalculate the trajectory

$$v_0 = \sqrt{\frac{\text{Range} \times g}{\sin(2\theta)}} \quad \leftarrow \textcircled{1}$$

```
[4] # Cell 4  
v0 = np.sqrt(rng * g / np.sin(2 * theta)) ← ②  
plot_trajectory()
```



Circus Performer Projectile Motion



Session 03 – Topics

- Learn about Python's **dictionary** data structure and how to read data files in **CSV** and **JSON** formats
- Analyze chemical trends and anomalies across the **Periodic Table of Elements**
- Implement the **linear regression** formulas to find the **line of best fit** using the method of least squares created by Gauss
- Identify an *unknown element* using the **Ideal gas law**
- Determine the height and velocity of **cosmic ray** showers using Newtonian Kinematics
- Simulate the **trajectory** of a circus cannon performer to determine a safe initial launch velocity